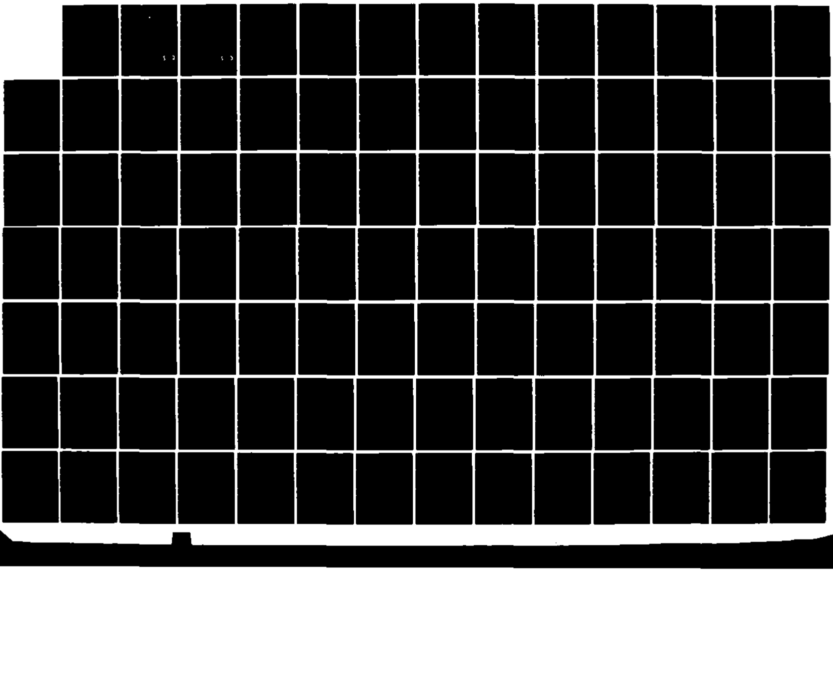
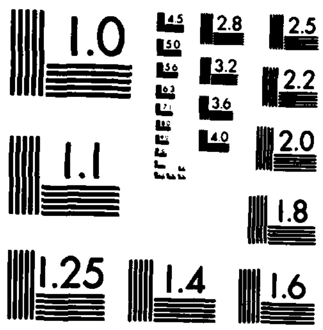
```
AD-A149 950   DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM              1/3
              PERFORMANCE MONITOR VOLUME 1(U) AIR FORCE INST OF TECH
              WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..   P D BAILOR
UNCLASSIFIED  DEC 83 AFIT/GCS/EE/83D-2-VOL-1              F/G 9/2     NL
```




1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AF INSTITUTE OF TECHNOLOGY

DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM PERFORMANCE MONITOR VOLUME I

THESIS

AFIT/GCS/EE/83D-2

PAUL D. BAILOR
Captain USAF

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/83D-2

DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM PERFORMANCE MONITOR VOLUME I

THESIS

AFIT/GCS/EE/83D-2

PAUL D. BAILOR
Captain USAF

Approved for public release; distribution unlimited

AFIT/GCS/EE/83D-2 -Vol

DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM PERFORMANCE MONITOR

VOLUME I

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Paul D. Bailor, B.S.

Captain USAF

Graduate Computer Systems

December 1983

Preface

The purpose of this study was to develop a performance monitor for a Data Base Management System. During the requirements analysis, many references on the subject where found, and they indicated the advantages of using such a monitor. However, none of the references contained a detailed examination of exactly what was measured and how the measurements where taken.

In this report, I have attempted to fill this gap by analyzing the performance evaluation process, providing tables of performance parameters, presenting a generalized design for a DBMS performance monitor, and presenting the details of implementing and using the monitor.

I would like to thank my advisor, Dr. Gary Lamont, and the other members of my thesis committee for their help in preparing this report. Additionally, I would like to thank Dr. Thomas Hartrum and Major Walter Seward for their excellent instruction in the areas of Computer Performance Evaluation and Queueing Theory. The information obtained in these classes was extremely helpful in preparing this report. Lastly, I wish to acknowledge my deep gratitude to my wife, Tammy, for her support and encouragement during the course of this study.

Paul D. Bailor

Contents

VOLUME I

List of Figures

List of Tables

AFIT/GCS/EE/83D-2

Abstract

thesis
This ~~study~~ focuses on the problem of evaluating the performance of a Data Base Management System (DBMS). In this study, DBMS performance evaluation is treated as a subset of computer performance evaluation, and in doing this, the performance parameters unique to a DBMS were developed and merged with the performance parameters associated with a general purpose computer system.

Based on this approach, a generalized design for a DBMS performance monitor was developed. This design emphasizes the use of existing performance tools such as software monitors and accounting packages, and it takes the performance monitoring requirements of different types of DBMS users into consideration. Additionally, the design is applicable to any type of DBMS regardless of the underlying data model.

The generalized design was implemented on a VAX 11/780 computer for the TOTAL DBMS. The results of the implementation showed the generalized design was viable and capable of measuring many different types of DBMSs. However, existing performance tools were only capable of providing a high level picture of DBMS performance. A specialized tool called an instrumentation utility had to be developed to gather detailed performance information.

I. Introduction

Background

A data base is a collection of stored, operational data, and a Data Base Management System (DBMS) performs the task of managing and manipulating the data contained within a data base (ref. 11:7-25). The development of generalized DBMSs in the late 1960s provided many advantages to the users of computer systems (ref. 34:3). However, these advantages were provided at a cost, and this cost is the overhead the DBMS places on the resources of a computer system. The amount of DBMS overhead and the quality of performance will vary based on the hardware configuration, the operating system characteristics, the architecture of the DBMS, and the structure of a data base query or update. Therefore, DBMS overhead should be measured to determine where changes can be made to help improve overall computer system and DBMS performance.

To effectively measure the variations and identify areas where changes can be made to reduce the overhead, a data base administrator (DBA), computer system manager, or software engineer needs the measurement and analysis capabilites provided by a performance monitor (ref. 2:315). Otherwise, experience, intuition, and trial and error techniques must be relied upon to identify areas for change and to achieve performance gains (ref. 48:1).

Consequently, a performance monitor is a much needed tool for the management of computer resources. In fact, a DBMS performance monitoring effort conducted at General Motors Corporation detected several system deficiencies during the installation process for the DBMS performance monitor. The correction of these deficiencies improved the performance of their REGIS DBMS by almost an order of magnitude before the performance monitor actually began collecting system data (ref. 39:331).

Purpose

The purpose of this study is to develop a methodology for conducting a performance monitoring effort on a DBMS and to develop a generalized design for a corresponding DBMS performance monitor. The development of these tools will allow the software engineer, data base administrator, and computer system manager to "easily" measure the performance of the DBMS and its effect on the supporting resources of the entire computer system. The data provided by these tools is used to evaluate DBMS performance based on the specific objectives of the DBMS users, and this evaluation provides a basis for identifying areas where changes can be made. The successful application of the changes should reflect an increase in the performance aspects of the DBMS and a reduction in the overhead effects on the supporting resources, thereby, allowing the DBMS users more flexibility in meeting their stated objectives.

Problem

At the highest level of abstraction, the general problem is to determine how to measure and analyze computer system resources so they can be used in the most efficient and effective manner. A study at this level would correspond to a performance evaluation effort of an entire computer system. At the next level of abstraction, the problem is to determine how to measure and analyze the performance characteristics of a specific computer system resource so it can be used in the most efficient way. Even though a DBMS can be considered a specific resource, it requires the use of most of the other resources of a computer system to provide its services to a user. Therefore, this study is concerned with both levels of abstraction, and a formal problem statement based on the two levels of abstraction is given below.

Problem Statement. The specific problem is to determine how to measure and analyze the performance characteristics of a DBMS and its supporting resources, and the objective of the solution is to identify areas where changes can be made to minimize DBMS overhead costs and increase overall performance.

Scope

This study will develop the requirements for a generalized DBMS performance monitor and generate an associated design. Every attempt will be made to design a DBMS performance monitor powerful enough to be used with a relational, hierarchical, or network model DBMS. However, this study will only implement the generalized design for the TOTAL DBMS (network model) developed by Cincom Systems Inc. (ref. 8:1-3). The TOTAL DBMS is currently being used on the Digital Equipment Corporation VAX 11/780 computer system in the Electrical Engineering Department of the Air Force Institute of Technology (AFIT).

Assumptions

1. A generalized design for a DBMS performance monitor is feasible. However, the specification of the system instrumentation required to connect the performance monitor to the measured system must be "open-ended" due to the wide variations in the implementations of computer system architecture.

2. The VMS operating system of the VAX 11/780 computer maintains tables reflecting the use of computer resources by programs executing on the computer (ref. 25:Appendix B). These tables are assumed to be accurate.

Approach

This problem was solved as a sequence of five logical stages and what was accomplished in each stage is outlined below. This sequence of stages is not necessarily a discrete or chronological sequence since areas of some stages overlapped with others.

Literature Search Stage. In this stage, the current literature on computer performance evaluation, DBMSs, and developing "user friendly" interfaces was reviewed. The goal of this stage was to provide the insight and additional background knowledge required to analyze the requirements for and design a generalized DBMS performance monitor.

System Analysis and Requirements Stage. In this stage, the requirements for a DBMS performance monitor were analyzed. The goal of the requirements analysis was to answer questions such as:

1. What type of user interface must be developed?
2. What parameters of a DBMS and computer system must be measured?
3. What type of monitoring technique should be used and how powerful should the monitor be?

4. What type of analysis must be performed on the recorded measurements?

The results of the system analysis were used to: specify the functional requirements of a generalized DBMS performance monitor, determine the performance tools required to measure DBMS performance, determine the tools/techniques used to analyze the measured performance data, and specify a test plan for the implemented DBMS performance monitor. Data Flow Diagrams (DFDs) and tables were used as much as possible to present a structured requirements analysis/specification technique.

System Design Stage. The goal of the system design stage was to develop a design document for a generalized DBMS performance monitor. Structured Analysis and Design Technique (SADT) diagrams were used as the design development tool, and the design document for this technique is presented in the form of a Reader's Kit.

Design Implementation Stage. In this stage, the generalized design was implemented for the TOTAL DBMS used on the VAX 11/780 computer system. The first step in the implementation was to study the details of the VAX 11/780 architecture and the TOTAL DBMS. A study of these details was necessary to completely specify the system instrumentation used to connect the monitor to the measured system. Once the details were determined and incorporated into the design, the second step was the actual coding and testing of the performance monitor.

Analysis/Validation Stage. In this stage, the generalized design and the implementation for the TOTAL DBMS and VAX 11/780 were evaluated for compliance with the functional requirements. As a result of the evaluation, deficiencies in the generalized design and implementation were identified, and recommendations for future improvements were made.

Sequence of Presentation

Chapter 2 of this study presents a complete analysis of the problem of conducting a performance evaluation effort on a DBMS. The corresponding requirements documentation, in the form of Data Flow Diagrams and Tables, is contained in Appendices A through C. Chapters 3, 4, and 5 concentrate on the design, implementation, and testing of a DBMS performance monitor. Chapter 3 contains the development of the generalized design for a DBMS performance monitor, and the corresponding documentation for this design is contained in Appendix D. Chapter 4 takes the generalized design and targets it for the TOTAL DBMS used on the VAX 11/780 computer system. The corresponding documentation pertaining to the details of the VAX 11/780 and the TOTAL DBMS are contained in Appendix E. Chapter 5 presents the program development steps and the results of testing the implementation against the stated functional requirements for a DBMS performance monitor. The program documentation is contained in Appendix F, and a Users Manual for the developed performance monitor is contained in Appendix G. Lastly, Chapter 6 contains the overall results and conclusions of this study as well as some recommendations for future study.

II. SYSTEM ANALYSIS AND REQUIREMENTS

Introduction

This chapter presents the system analysis performed on the problem of monitoring Data Base Management System (DBMS) performance. Based on this analysis, the functional requirements for a DBMS performance monitor are presented. To meet the needs of all potential readers, this chapter is presented in two parts. Part One contains background information on DBMSs and Computer Performance Evaluation (CPE) concepts, and it is intended for readers either unfamiliar with these concepts or wishing to review them. All other readers should skip to Part Two which contains the system analysis discussion and presents the functional requirements derived from the system analysis.

PART ONE - BACKGROUND INFORMATION

A requirement for the analysis of any system is a common understanding of system definitions (ref. 50:7-23 and 41:34-36). This section defines the concepts and terminology associated with DBMSs and CPE. The background information on DBMSs is presented first followed by the CPE information. All information is introductory in nature; however, it represents the "core" concepts and terms associated with DBMSs and CPE. Since this information is used throughout the study, lists of the terms and concepts are provided to serve as a reference aid if they need to be reviewed again.

DBMS Definitions

The following DBMS terms and concepts are defined:

1. DBMS Data Models.
2. Data Base Schema.
3. DBMS Architecture.
4. DBMS Environment.
5. DBMS - Computer Architecture Semantic Gap.
6. Computer/DBMS Boundaries.

DBMS Data Models. A DBMS uses a data model as its underlying structure. The data model serves as a basis for data definition and manipulation languages because it defines the data structures and associated operators (ref. 2:Chapter 4; 10:63-73; and 49:Chapter 3). The three best known data models are listed below with a brief description of how the data is represented. In addition to the three data models, some DBMSs are based on the technique of file inversion on multiple keys; however, this is more of a physical implementation than an abstract data model (ref. 2:83-84).

1. Relational - the data is represented as tables which are a special case of the mathematical construct known as a relation.

2. Hierarchial - The data is represented by tree structures.

3. Network - The data is represented by records and links.

Data Base Schema. To implement a DBMS, the abstract data models need to be translated from a model into an operational system. In order to perform the translation, the model needs to be described in a form suitable for implementation, and this description is called a schema (ref. 51:368 and 11:22-23). Therefore, a schema is a collection of information describing the data base, and it provides the necessary mechanisms for the data base objective of data independance. An important part of a schema is the description of the data elements contained in the data base. This description is used to store data element values into the proper position of data base files and to locate data element values. Additionally, it is possible to describe only a part of a data base object/record, and this type of description is known as a subschema.

DBMS Architecture. Date defines a DBMS architecture to consist of three levels: the external level, the conceptual level, and the internal level (ref. 11:17-19). Figure II-1 contains a diagram of the architectural model, and brief descriptions of each level of the model are:

External Level. This is the level closest to the end-users of the data in the data base, and it is concerned with the way data is viewed by each user.

Internal Level. This level is the one closest to physical storage, and it is concerned with the way data is actually stored. It is responsible for the handling of stored data base records, and it provides the stored record interface to the access method which is used to retrieve and store physical data base records. The access method is not a part of the DBMS. It deals with the hardware, device-dependent details of physical storage, thereby, concealing these details from the DBMS.

Conceptual Level. This level is the connection between the other two levels, and it is concerned with the global view of data contained in the data base(s). In other words, the conceptual level provides a view of the entire data base while the external

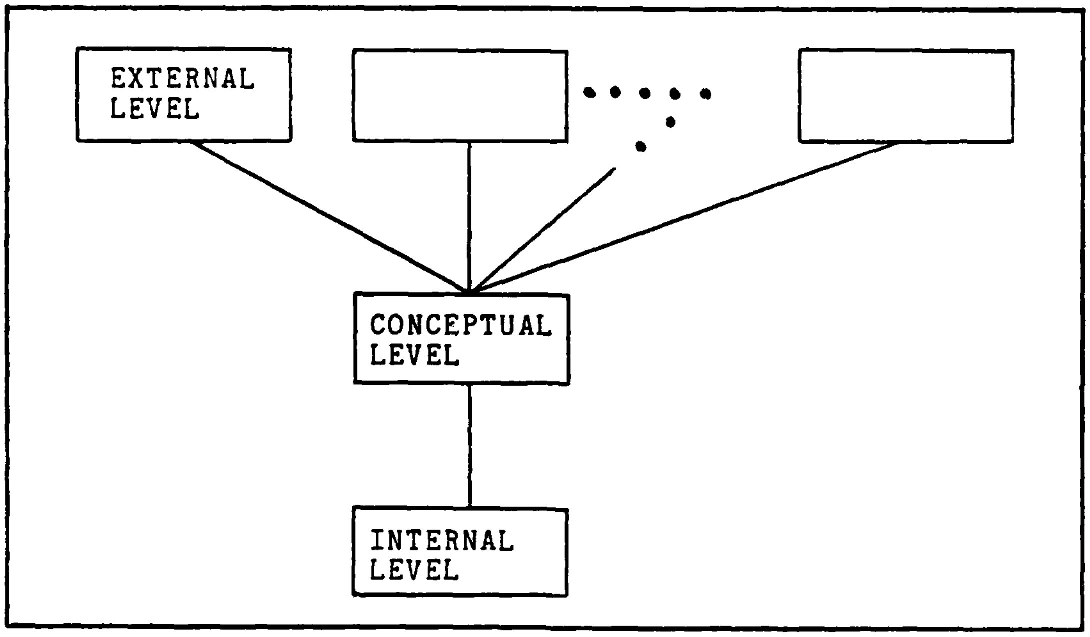


Figure II-1. DBMS Architectural Model

level provides the end-user with a view of some portion of the entire data base.

DBMS Environment. The DBMS environment is composed of two components. The first component is the type of DBMS, and the second component is the workload characteristics of the DBMS. The type of DBMS defines the implementation details, and a thorough understanding of the implementation and workload details is important to the performance evaluation of any system (ref. 5:6). Exactly why these details are important is covered later in the Methodology/Procedures section of Part Two of this chapter. At this point, it is only important to introduce high level definitions for the two components of DBMS environment because a common, high level understanding is necessary before the details can be presented.

Based on a review of the current literature, three types of DBMSs can be defined, and their definitions are:

1. Single Machine (Conventional) DBMS - This type of DBMS is defined to be a DBMS composed entirely of software modules and implemented on a single, Von Neumann architecture, general purpose, computer (ref. 34:4-5). It is referred to as a conventional DBMS because it was the first type to be developed, and it is the most commonly used of the three DBMS types (ref. 10:). (Note: The design of today's general purpose computers is commonly referred to as the Von Neumann architecture because they are based on the Von Neumann architectural model derived in the 1940s. This study also uses the Von Neumann classification to describe the architectural design of computers; however, this classification should not obscure the Analytical Engine designed by Babbage in the 1840s which also embodied many of the design principles of today's computers.)

2. Data Base Computer - This type of DBMS is defined to be a DBMS implemented on an architecture whose only purpose is to perform data base functions and tasks. In general, this means the data base functions have been

offloaded from a host computer and moved onto a dedicated data base computer which is directly connected to the host computer by a communications link. The data base computer can be implemented in one of two ways. First, it can be a general-purpose, Von Neumann machine using a DBMS composed entirely of software modules, and this configuration is commonly called a backend data base management system (ref. 34:3-4). The second implementation is a special purpose machine where the data base functions are wholly or partially implemented in hardware (ref. 6:Chapter 1).

3. Distributed DBMS - This type of DBMS is defined to be a network of computers where nodes within the network maintain a DBMS (ref. 47:440-441). The actual data base contents at the nodes of the network could be copies or partitions of a common data base or entirely different data bases. The architecture of the DBMSs at the nodes of the network could be either conventional or Data Base Computers.

The second component of DBMS environment, DBMS workload characteristics, varies from computer installation to computer installation; however, the workload for any given type of DBMS can be grouped into two generalized classes defined by Hawthorn and Stonebraker (ref 29:3). The first class is the "overhead intensive workload" which is defined to be that workload for which the data processing time is less than system (operating system and data management) overhead to process the workload. The second class is the "data intensive workload" which is defined to be that workload for which the time to process the data is much greater than the overhead.

DBMS - Computer Architecture Semantic Gap. Perhaps the largest contributor to DBMS performance problems is the incompatibility of DBMS functional requirements with the architecture of a general purpose computer (ref. 6:1 and 36:422). A general purpose, Von Neumann architecture computer uses the addresses of storage locations to retrieve and store data values. On the other hand, the functional requirements of a DBMS are oriented toward retreiving and storing data based on data values without regard to the address of the storage location. Therefore, a significant part of the processing time associated with a conventional DBMS is attributable to the conversion between the data referencing schemes of address-by-location of the computer and address-by-value of the DBMS.

Computer/DBMS Boundaries. By definition, a boundary is something that indicates or fixes a limit such as a separating line. One boundary in a computer system is the boundary between the system workload and the system resources. At first glance, this may appear to be an easily definable boundary, but further investigation shows it is not. There are three types of resources involved with a computer system, and they are human, hardware, and software. Hardware resources are easily defined by a configuration diagram; however, human and software resources are not as straight-forward. Operators of a computer system are a human resource as necessary as the hardware resources, but operators can also contribute both negatively and positively to the workload based on their knowledge, efficiency, and ability to operate the system. Human users are external to the computer system; therefore, they contribute to the system workload through user jobs.

Software resources such as operating system modules are necessary to control the system and are usually considered a resource of the system and not a part of the workload. On the other hand, software resources such as compilers, text editors, DBMSs, and other types of shared programs and data perform work as needed by users of the computer system. Therefore, they contribute to the system workload based on the individual needs of the users, but at the same time, they can be considered a system resource since they are available to all users of the system to preclude the users need for developing their own (ref. 26:1-2; 33:8; and 43:13).

As the preceding paragraph shows, a DBMS could be included in either one or both sides of the system resource - system workload boundary. Ferrari (ref. 24:4) states the system's boundaries must be specified clearly once the objectives of the performance study have been clarified; otherwise, an inconsistent interpretation of the boundaries may be used for the performance study!

Based on this study's objectives of providing performance information to both data base users and the management of the computer system, the following specification of the computer/DBMS boundary is used for the remainder of this study.

Computer System. The DBMS (as well as all other system software), all supporting and applications programs, and all the data bases are part of the system workload (ref. 24:221-222; 26:1-2; 33:8; and 43:13).

DBMS User. The DBMS, DBMS support programs, and all data bases are considered a system resource. Applications programs, using the DBMS as a system resource, are a part of the DBMS workload which is a subset of the overall system workload (ref. 24:221-225).

CPE Definitions

The following CPE terms and concepts are defined:

1. Performance Index.
2. "Acceptable Level" of Performance.
3. Types of Performance Monitors.
4. System Instrumentation.
5. Performance Monitor Artifact and Accuracy.
6. Performance Monitor Power.
7. Collection of Performance Measurement Data.

Performance Index. A performance index is a descriptor used to represent a system's performance (ref. 24:11). Ferrari (ref. 24:12-13) lists the three most popular classes of quantitative performance indices as productivity, responsiveness, and utilization. These indices can be further grouped based on two categories of measures proposed by Svobodova (ref. 46:14-15). The first category is effectiveness which is defined in terms of the system's capability to process a given workload and to meet the time requirements of individual users. The second category is efficiency, and it is defined as the internal delays and utilizations of individual system components versus demand. Effectiveness and efficiency serve as excellent categories for performance indices because they are also the two major goals of engineering.

Based on the work by Ferrari, Svobodova, and this study, the scheme for representing a set of performance parameters based on performance index category and class is shown in Figure II-2. The performance indices provided by Ferrari have been expanded to take into consideration the integrity and security aspects of computer system effectiveness, and the efficiency aspects have been further expanded to include the allocation and deallocation of resources. The performance indices were expanded to more accurately characterize the operation of a computer system, and this area is covered in more detail in Part Two of this chapter. Additionally, Part Two contains a breakdown of the performance indices into quantifiable performance measures or parameters.

"_Acceptable Level_" _of Performance_. An acceptable level of performance is easily defined once the objectives of the end-users and computer system management are clearly defined. These objectives are required to formulate a set of performance criteria to which corresponding performance measurements can be compared. This basis for comparision makes it possible to decide whether or not performance is acceptable, and it allows the effects of changes to be measured - a necessary prerequisite for any optimization process (ref. 35:2).

SYSTEM EFFECTIVENESS
- -Productivity

  .

  .

  <Performance Parameters>

  .

  .
- -Responsiveness
- -Integrity
- -Security

SYSTEM EFFICIENCY
- -Allocation
- -Utilization
- -Deallocation

Figure II-2. System Performance Indices

Generally speaking, the level of performance is the degree to which the computer meets the expectations of a user (ref. 46:8). However, this definition of performance leads to a trap which must be avoided. The trap is the objectives and performance criteria ("expectations") established by the user may exceed the capabilities of the computer in question. Therefore, a different definition of performance may want to exclude the users expectations, and this definition would only consider the effectiveness with which the resources of a computer system are utilized. Singularly, neither of these definitions adequately describe a level of performance. Hence, both of them must be used to describe the effectiveness with which the users objectives are met (system effectiveness), and the effectiveness with which the available resources are utilized (system efficiency). In other words, the system effectiveness must be weighed against the system efficiency in establishing an "acceptable level" of performance (ref. 46:8-9).

<u>Types of Performance Monitors</u>. There are three basic types of performance monitors, and they are (ref. 46:Chapter 6 and 24:29-64):

1. Software Monitor - A software monitor is a special program contained within the measured system, and its purpose is to collect information about system processing and utilization of system resources. The collection process is usually driven by one of two techniques. The first technique is an event-driven monitor, and this type of monitor is activated when some type of event (e.g. I/O interrupt, expiration of a CPU time quantum, arrival of a new job, etc.) occurs within the system. The second technique is a time-driven monitor, and this type of monitor is periodically activated when a specified time interval has expired. When software monitors are activated, they are given control of the system to measure and record values for the necessary performance parameters, and when the measurement process is completed, they return control to the operating system. Of the two collection techniques event-driven monitors

record exact data; whereas, time-driven monitors record a large number of samples to allow for statistical inferences to be made.

Software monitors can take several forms, and a common form found on most computer systems is a job accounting package. Other forms of software monitors are: specially designed programs that interact with the operating system and applications programs that contain special instrumentation statements for measuring and recording events within the program.

2. Hardware Monitor - A hardware monitor is a free standing device used to sense electronic signals (hardware events) within the circuitry of the computer and record the information in the memory of the computer system. A special case of a hardware monitor is a monitor incorporated into the micro-instructions of a computer system, and this type of monitor is commonly called a firmware monitor.

3. Hybrid Monitor - A hybrid monitor is a combination of software and hardware performance monitors that interact to collect performance measurements. The software part of the hybrid monitor can detect and record software related events as well as generate signals that can be detected and recorded by the hardware part of the monitor. The hardware part of the hybrid monitor combines signals generated by the software monitor with the

hardware events it detects to formulate different types of measurement data.

System Instrumentation. Instrumentation is the facility used to connect a performance monitor to the measured system allowing a set of system activities to be observed. System instrumentation can take many forms. For example, hardware monitors typically use electronic probes to connect to the electronic circuits within the computer hardware, and an event-driven software monitor for an interrupt driven machine could be instrumented by a special data collection routine that executes prior to or after the interrupt service routine.

The instrumentation of a performance monitor is one of its most important aspects; however, it is also the most difficult to specify. While today's general purpose computers have a common architectural design, the implementation of this common design varies widely between computer vendors. Additionally, the common design for computers does not address measurement facilities and only a few computer vendors have done anything to solve this design problem (ref. 24:46). Therefore, the specification and design of the instrumentation for a particular type of computer system is performed only after the machine has been built, and this leads to a non-generalized implementation of system instrumentation even though all general purpose computers have the same basic architectural design.

The preceding paragraph points out one of the biggest problems with performance monitors - the concepts of performance monitors are common; however, a universal tool is almost impossible to build because of system instrumentation problems and the many different implementations of a common architectural design. System instrumentation also causes another type of problem, and this problem deals with the user interface to performance monitors. For example, it may require a detailed knowledge of the system to determine where to connect the electronic probes of a hardware monitor, and the connection process may contain hazards such as electrical shock and device damage. The installation of an event-driven software monitor may require modifications to the operating system which also requires a detailed knowledge of the system.

The problems listed above are reflected in the results of this study. The specification of the system instrumentation for a DBMS performance monitor is very generalized in the system design chapter (Chapter 3), and the system instrumentation was not completely specified until the VAX 11/780 and TOTAL DBMS implementation details had been examined.

Performance Monitor Artifact and Accuracy. The term monitor artifact is used to describe the way a performance monitor alters or interferes with the normal operation of the system (ref. 46:82). For example, a software monitor must compete for system resources just like user jobs within the system; therefore, the monitor effects the processing characteristics of the computer system and the DBMS. The artifact of performance monitors should be kept as low as possible, but depending on how complete a picture of performance is desired, the monitor artifact can produce noticeable side effects (ref. 24:29). This is another of the many reasons for establishing objectives. The performance monitoring objectives will dictate what type of performance information must be measured and only necessary information should be measured. Anything else will add to the artifact of the monitor and provide little additional data to the performance monitoring study.

Generally speaking, a hardware monitor presents the smallest amount of artifact or interference to the measured system, and for this reason, they are usually considered to be more accurate than software monitors (ref. 24:45). Software monitors may substantially interfere with the measured system, and they can detect only the more macroscopic, less frequent events (even though there are some types of events that can be detected by both hardware and software monitors). Therefore, software monitors generally provide a lower degree of resolution and accuracy than a hardware monitor does.

Performance Monitor Power (ref. 46:88-89). The power of a performance monitor is determined by the monitoring technique and the actual implementation of the monitor. Together with monitor artifact, monitor power can be used as the criteria for the design, evaluation, and selection of performance monitors. The five dimensions of monitor power are given below:

1. Monitor Domain - Monitor domain is the class of activities theoretically observable with a particular monitoring technique. Note the difference between monitor domain and instrumentation - Instrumentation facilitates application of a monitoring technique to a particular problem; it selects a unique set of measurable events from the monitor domain.

2. Input Rate - Input rate is the maximum frequency at which events can be recognized and recorded.

3. Input Width - Input width is the number of bits of input information the monitor can extract and process when a monitored event occurs.

4. Recording Capacity - Recording capacity is the number of memory elements that are available for storing extracted information, and it determines the

amount of information that can be retained for further processing.

5. Monitor Resolution - Monitor resolution is the resolution of the time clock from which the monitor derives timing information. This factor limits the achievable accuracy of time-based measures.

Collection of Performance Measurement Data. The collection of performance measurement data is rather straightforward since it is performed by the performance monitor; however, choosing or designing the type of performance monitor to use is a more difficult task. Ideally, the chosen or designed performance monitor will measure values for the necessary performance parameters, provide accurate results, be easy to use, and produce a minimum amount of artifact (system interference). Unfortunately, the ideal performance monitor is rarely available because of the problems associated with instrumenting a given computer system to allow performance measurements to be taken. Therefore, several performance monitors may need to be examined to decide which one or ones provides the best service.

For example, the domains of four different types of commercially available performance monitoring tools are presented in Table II-1, and this table illustrates the differences in the domains of the the four

TABLE II-1

Domains of Performance Measurement Tools

Software Monitor - Generally speaking, a software monitor can measure operating system events and those hardware events that transfer control to a specified location (interrupts). There are some system activities that can be measured both by a software and a hardware monitor; however, the domains of the two types of monitors do differ significantly.

Hardware Monitor - A hardware monitor measures the electronic control signals within the system that are used for the low level communications between system resources and components. Therefore, it is very useful for measuring component activity and overlap, but since it is a passive device, it cannot associate the activity with a particular software process unless the software process generates a special control signal for the hardware monitor. Additionally, the hardware monitor can be used to decode and measure the contents of system registers or memory addresses. For example this capability is useful for monitoring instruction mixes.

Job Accounting Package - Job accounting packages will provide good data on the workload and processing characteristics of individual user jobs as well as generalized information on system processing characteristics such as turnaround and response times. By performing statistical analysis on this data, generalized inferences about the overall computer workload and system processing can be made.

Instrumented Program - The domain of an instrumented program is the same as that for a software monitor; however, an instrumented program will narrow the scope of the measurements to obtain more specific information about a particular software task as opposed to the more generalized, overall scope of a software monitor.

Calculated - Calculating performance measurement values extends the domain of all the different types of performance monitors since the calculation process uses two or more values measured by a performance monitor to de~ive a value not otherwise directly measurable.

different monitors. Additionally, Table C-4 of Appendix C presents a complete comparison of several different performance monitors. Table C-4 contains information on the domain, accuracy, artifact, and the advantages/disadvantages of the different types of performance monitoring tools.

Summary of Part One

This part of Chapter 2 presented an introduction/review of the common concepts and terminology associated with DBMSs and CPE. An understanding of this background information is essential to the system analysis of the DBMS performance monitoring process which is presented next, as Part Two of this chapter.

PART TWO - SYSTEM ANALYSIS

This part of Chapter 2 contains the system analysis applied to the problem of developing a DBMS performance monitor. This analysis includes: the specification technique used, the performance evaluation process, the system requirements, the performance monitoring requirements, and the methodology/procedures for conducting a performance study of a DBMS. After the analysis stage has been completed, the functional requirements for a generalized DBMS performance monitor are extracted. Chapters 3, 4, and 5 use the functional requirements as the baseline for designing, implementing, and testing a DBMS performance monitor.

Overview of Previous Studies

A literature review of Data Base Management Systems (DBMSs) resulted in a varied assortment of books, reports, and articles related to the performance evaluation of a DBMS. These studies fell into three basic categories:

1. Design issues and design modelling tools for increasing DBMS performance.
2. Analytic studies of a specific DBMS or DBMS architecture.
3. Performance evaluations of a specific DBMS.

Each of these studies contributed valuable research conclusions, and the results of the studies in the third category provided useful information for this study. However, the studies did not contain a complete approach to the problem of DBMS performance evaluation. Specifically, studies in the third category concentrated on presenting the results of evaluating DBMS performance, but they contained minimal information on the set of performance parameters measured and the performance tools used to measure values for the parameters.

Based on the literature review, a detailed analysis of the DBMS performance evaluation process needed to be conducted. A detailed analysis of this process was not originally anticipated as a part of this study; however, it is a gap that needed to be filled. The remainder of this chapter fills this gap by: analyzing the DBMS performance evaluation process, developing the set of performance parameters used to characterize DBMS performance, and determining the types of performance tools necessary to measure values for the set of performance parameters.

Specification Technique

To begin the detailed analysis, the performance evalution process in general was examined, and Data Flow Diagrams (DFDs) were used as the structured analysis tool for defining and specifying this process. This technique was used because DFDs approach a situation from the point of view of the data, and they can serve as a simple model of the real situation (ref. 14:40-41). In performance monitoring, the measurement data is the driving factor, and without a model of how the measurement data is transformed and used, the following situation could easily occur - A performance monitoring tool is used to measure and accumulate performance data, and after several samples of data are collected, the user wonders what to do with the data (ref. 5:5).

Data Flow Diagrams. The mechanics of a DFD diagram are shown in Figure II-3. An input data source provides data to a data transformation process. The transformation process converts the input data into ouput data which is provided to the data sink. A transform process can have more than one input data flow and produce more than one output data flow. Additionally, the transform process can access data files or data bases while performing the data transformation process.

Data Flow Diagrams are intended to show the steady state flow of data within a system with no consideration to control paths such as loops; hence, loops appear very seldomly in DFDs. One situation where loops may occur is an iterative testing process such as hypothesis testing. In these situations, data and test conditions can be

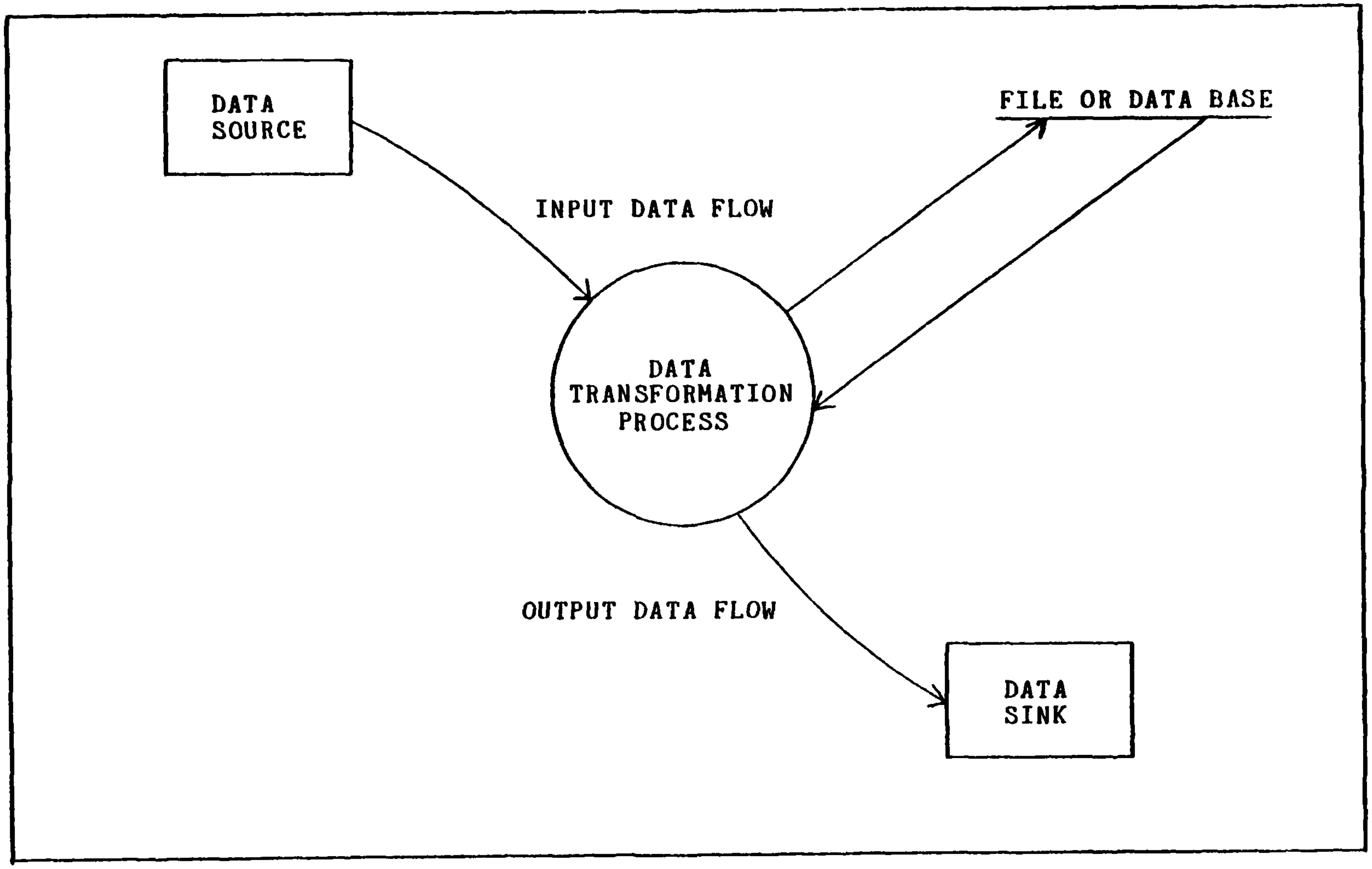


Figure II-3. Data Flow Diagram (DFD)

modified several times to give a broader range of results. Showing data flow through a hypothesis testing process would naturally seem to require some indication of an iterative process or loop. Since loops are seldomly contained in DFDs, there is no agreed upon convention for showing a loop. For this study, a data flow constructed of dashed lines is used to show a loop in the data flow paths, and in keeping with the intent of DFDs, this convention is used only when absolutely necessary.

Performance Evaluation Process

A complete set of documentation for the process of computer system and DBMS performance evaluation was developed as a part of this study. This set of documentation includes indexes for the DFDs, the actual DFDs, and data dictionaries. The complete set of documentation is contained in Appendix A, and this chapter presents and discusses some of the diagrams contained in Appendix A. The diagrams presented in this chapter provide a general picture of the performance evaluation process with the details being reserved for the data dictionaries of Appendix A. The DFDs presented in this chapter and in the appendix are based on the information contained in the excellent report by Bell, et. al. (ref. 5:).

The high level concept of monitoring computer system performance is shown in Figure II-4. In this diagram, the

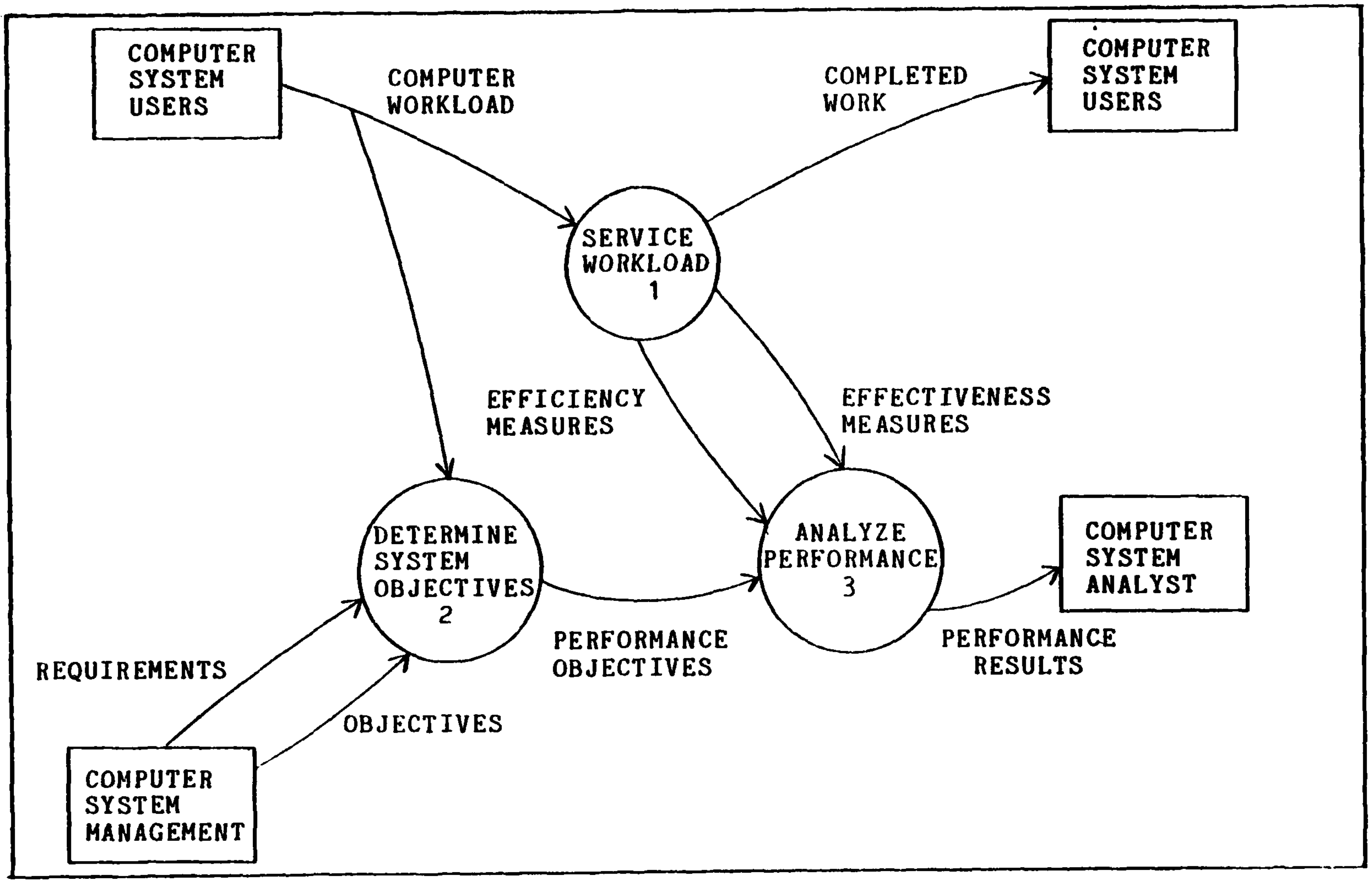


Figure II-4. Computer Performance Evaluation

users of the computer system generate the system input and receive the system output. The input is whatever task or job the user needs the computer to accomplish - the computer's workload. The output is whatever results the user requested the computer to produce - the completed work. The Service Workload process is the process that takes the input workload and produces the completed work, and the Service Workload process can be measured to determine its effectiveness and efficiency (the major categories for performance indices).

The parts of the diagram described above are the basis for establishing the performance indices of a computer system. The remaining parts of the diagram describe the process of establishing performance objectives and criteria used as the basis for measuring, changing, and optimizing the computer system. The Determine System Objectives process is an important process, and it evaluates the workload requirements of the system user (which implicitly includes the objectives of the user), the management requirements of the computer system management, and the system objectives of the computer system management to produce a set of performance objectives. The Analyze Performance process uses the performance objectives, effectiveness measures, and efficiency measures to produce a set of performance results. The performance results are used by a computer system analyst to determine how well the computer system is working, and if necessary, to identify areas where changes or modifications need to be made.

Of the processes contained in Figure II-4, process 2 (Determine System Objectives) and process 3 (Analyze Performance) have been explained in detail in the report by Bell, et. al. (ref. 5:). However, process one, the service workload process, is much less well explained and defined, especially in the area of DBMS performance evaluation. Based on this, the service workload process is examined in detail in the System Requirements section, but processes 2 and 3 are only briefly examined and explained in the Methodology/Procedures section of this chapter.

To establish a starting point, DBMS performance evaluation is defined as a subset of computer system performance evaluation, and this subset is shown in Figure II-5. Some type of DBMS workload in terms of interactive queries, applications programs, report writers, etc. is applied to the DBMS. The computer system(s) and DBMS are monitored to measure the effectiveness and efficiency of the system resources during the Service DBMS Workload process. Therefore, the performance indices for a DBMS are the same as the performance indices of a computer system; however, some of the specific performance measures defined beneath the performance indices may be different. The

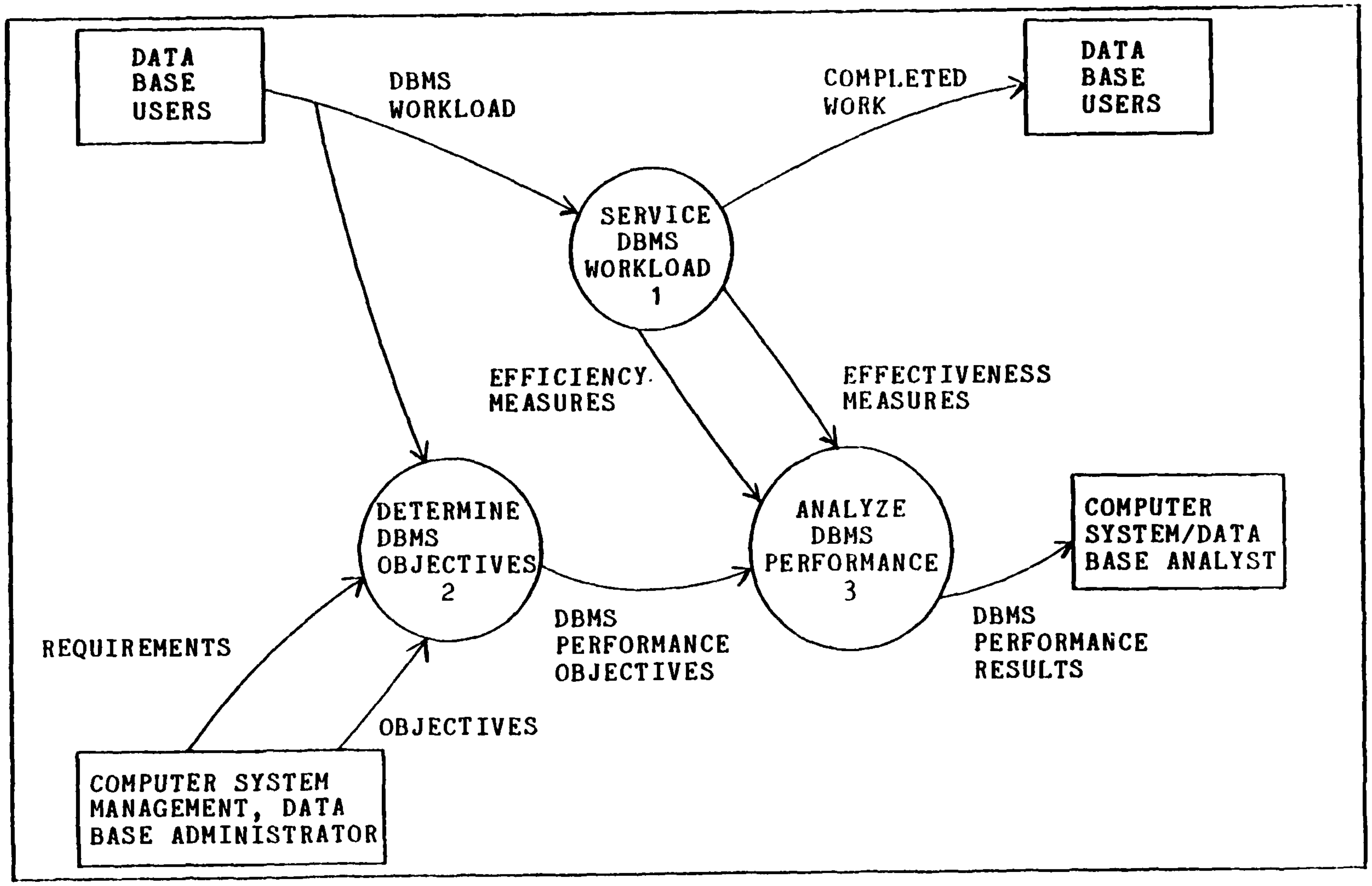


Figure II-5. DBMS Performance Evaluation

performance objectives for the DBMS are established in a manner similar to the way performance objectives for the computer system are established. The DBMS workload of the data base users (which implicitly includes the objectives of the data base users), the management requirements of the computer system management and data base administrator (DBA), and the objectives of the computer system management and DBA are inputs to the Determine DBMS Objectives process to produce a set of DBMS performance objectives. The Analyze DBMS Performance process uses the performance objectives, effectiveness measures, and efficiency measures of the DBMS to produce a set of DBMS performance results. The software engineer, DBA, and the computer system manager use these results to determine how well the DBMS is performing, and if necessary, to identify areas where changes or modifications need to be made.

While both concepts of performance evaluation appear to be straight-forward, the question of how to begin and carry out a performance evaluation can be non-trivial, especially if approached incorrectly (ref. 5:5-9). A seemingly simple question, but one with a large impact is - why would anyone want to evaluate DBMS performance? The answer to this question must lie in the set of clearly defined objectives for the computer system and DBMS. As stated in the definition of "acceptable level" of performance in Part One of this chapter, a set of objectives is required; otherwise, it is useless to monitor the performance of a computer system or DBMS since there is nothing to compare the performance measurements against. For example, the monitoring of a particular set of data base updates may show this task to require three hours of CPU time to complete, and without some objective to be met, an evaluation of whether or not the three hours of CPU time is a good or bad level of performance cannot be made.

Operational Objectives. Operational objectives in terms of this study fall into three categories. The first category is the operational objectives of the overall computer system which includes the DBMS and its corresponding data bases. The second category is the operational objectives of just the DBMS and its data bases. These two categories of objectives are established by the computer installation management, the DBA, and the end-users. Since the management and end-user objectives of every computer installation are different, this study can do nothing more than emphasize the importance of establishing operational objectives for the overall computer system. The operational objectives of the DBMS and its data bases also vary at each computer installation. However, generalized performance measurement objectives for each type of DBMS user can be defined, and these objectives are developed in the next section of this chapter. The third category of operational objectives, those of a DBMS performance monitor, can be generalized enough to apply to all users of the monitor. The four objectives of a DBMS performance monitor are defined below.

1. To provide the user of the DBMS with a measure of the resources required by the DBMS to accomplish a given data base task.

2. To allow the user to identify areas where a bottleneck may exist such that the user can make changes to the system to help alleviate or cure the bottleneck.

3. To provide the user with a convenient, "user friendly" performance monitoring technique for the evaluation of DBMS processing.

4. To minimize the artifact inherent in performance monitoring techniques.

It is important to note the objectives of the DBMS performance monitor are directed at a user. In this light, the term user must be studied very closely to determine if there are different levels of observation at which a user may "see" the DBMS. An architectural model of a DBMS is useful for relating the different DBMS users with an appropriate level of DBMS observation, and this model allows the performance aspects of interest to specific or all users of the DBMS to be defined.

User Relationship to DBMS Architecture. Using the DBMS architecture model defined by Date (ref. 11:17-19), two interfaces can be defined, and these interfaces are between the external-conceptual level and the internal level - access method. Each interface corresponds to a particular type of DBMS user and Figure II-6 lists these users. These users may or may not have the same overall objectives, and this impacts on the performance aspects they are interested in. In the following paragraphs, the two interfaces are studied in more detail to obtain a general idea of the performance aspects (in terms of space and time) that may be of interest to the different users.

External-Conceptual Interface. There is a "level of observation" both above and below this interface. Above this interface are the end-users of the data base with their corresponding local view of the data base contents and organization. Below this interface is the Data Base Administrator (DBA) who has a global view of the data base contents and organization.

An end-user can be further specified as either technically or non-technically oriented. A non-technically oriented end-user is normally associated

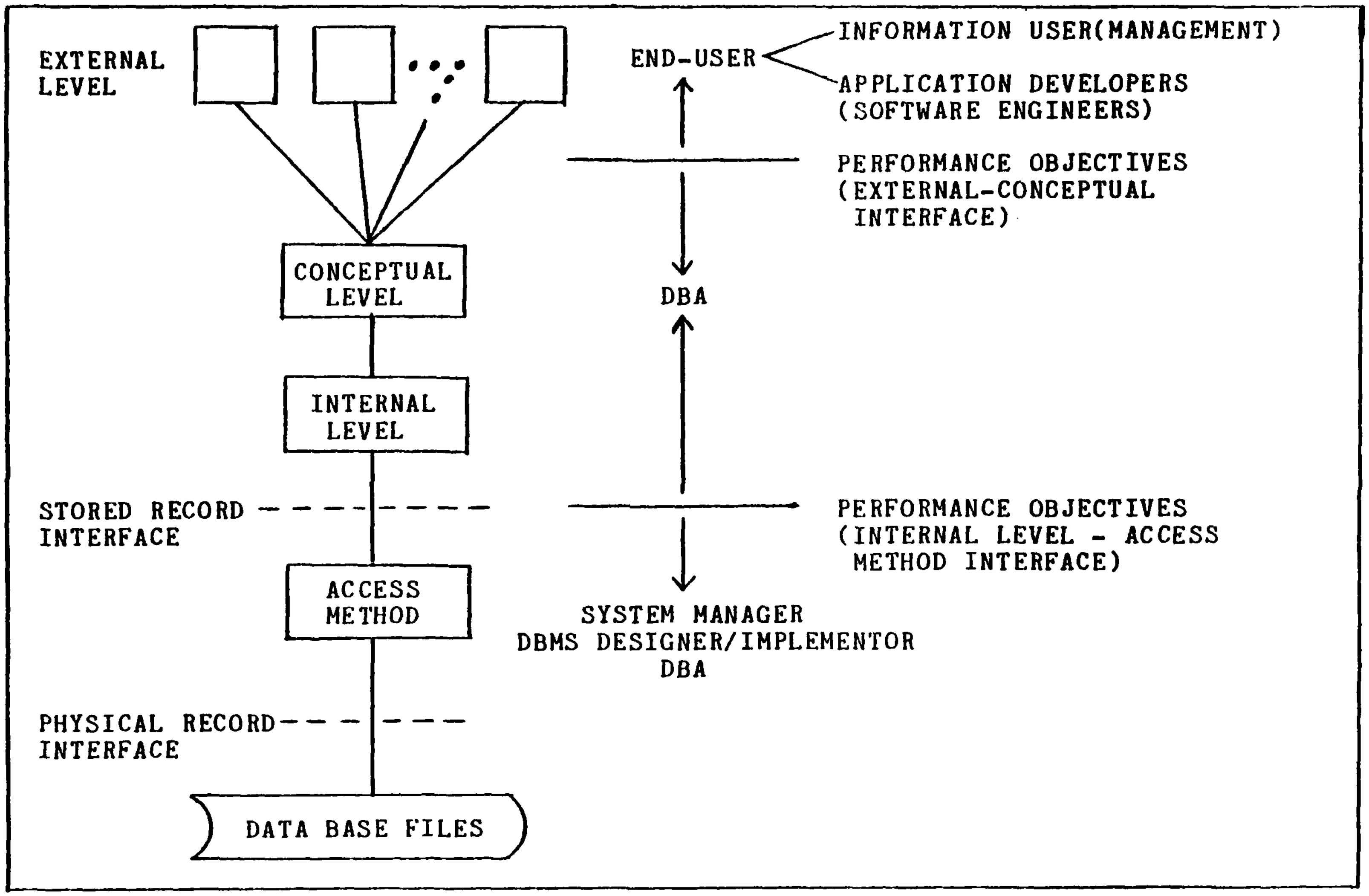

EXTERNAL LEVEL
CONCEPTUAL LEVEL
INTERNAL LEVEL
STORED RECORD INTERFACE
ACCESS METHOD
PHYSICAL RECORD INTERFACE
DATA BASE FILES
END-USER
INFORMATION USER(MANAGEMENT)
APPLICATION DEVELOPERS (SOFTWARE ENGINEERS)
PERFORMANCE OBJECTIVES (EXTERNAL-CONCEPTUAL INTERFACE)
DBA
PERFORMANCE OBJECTIVES (INTERNAL LEVEL - ACCESS METHOD INTERFACE)
SYSTEM MANAGER
DBMS DESIGNER/IMPLEMENTOR
DBA

Figure II-6. Performance Objectives of DBMS Users versus DBMS Architecture

with the management of the organization, and this type of end-user uses the computer and data base as a tool to access management information for supporting management decisions. This end-user normally accesses the data base through a query language or a "canned" application program and has one expectation of the system. This expectation is to be able to access the required information in a timely manner. How this impacts on the computer resources is not a concern unless the system doesn't respond within the time frame the end-user expects it to. Therefore, this type of end-user has a generalized performance measurement objective of determining if the desired data is accessed in a timely manner.

On the other hand, a technically oriented end-user corresponds to a software engineer who has been tasked by management to develop an information processing capability requiring access to some portion or to all of the data base. This user is concerned with both space and time requirements. Time is of primary importance so the results can be provided quickly, and space is important so the application being developed can be performed within the constraints of the system resources. This user typically needs relevant information about system parameters to assist in trade-off decisions (For example, more main memory space could be used to help reduce I/O wait times). Therefore, this user may require performance information on main memory, buffer space, file size, file distrubution, access paths, I/O time requirements, CPU time requirements, etc., pertinent to the specific development task. Without this type of information, the trade-off decisions can not be made except by trial and error. Based on the preceding discussion, this type of end-user has a <u>generalized performance measurement objective</u> of determining how to design and implement a solution that provides timely results as well as placing a minimum impact on the resources of the computer system and DBMS.

The DBA is a technically oriented user of the data base, but the DBA also has the responsibility for the overall control of the data base (ref. 11:25 and 2:29-39). The data base organization must allow all end-users to access the information they require, and it must use the system resources in an efficient manner. Hence, the DBA's task is comparable to the technically oriented end-user in that both of them require system information allowing them to make trade-off decisions. At this level of observation, the DBA will require information on how efficiently the end-users can gain access to the desired information contained in the data base(s). Therefore, the <u>generalized performance measurement objective</u> of the DBA at this level of observation is to determine how to design the data base organization, access paths, and schemas to allow the end-users to access the desired information while placing minimum overhead on the computer system and DBMS resources.

Internal Level - Access Method Interface. Again, there is a level of observation both above and below this interface. Above this interface is the DBA, and at this level of observation, the DBA must be concerned with the mapping of stored records at the internal level to the data base schemas at the conceptual level. In particular, the DBA must be able to evaluate the effect of the length of access paths, the time required to make the data in a stored record available to the end-user, and the amount of CPU processing required to process the indexes at the internal level of the DBMS. Therefore, the generalized performance measurement objective of the DBA at this level of observation is to determine how to adjust implementation and data base generation parameters to optimize the efficiency of the interface between the internal level of the DBMS and the access method of the computer system.

Below this interface, there are several observers of DBMS performance aspects. The first observer is the system manager, and the system manager must be concerned with the impact of the DBMS on the other resources of the computer system. Therefore, the system manager needs information on the percentages of the total resources consumed by the DBMS to process the DBMS workload versus the percentages of total resources consumed to process the overall computer workload. In other words, the system manager needs a concise picture of overall resource utilization and what percentage of the overall utilization is attributable to the DBMS. The system manager's generalized performance measurement objective is to determine if the existing system resources are providing an acceptable level of service to the system users as well as to find specific system bottlenecks or resource under-utilization. The second observer, the DBMS designer/implementor, has the task of mapping the functional requirements of the DBMS to the underlying architecture of the computer system. Therefore, the trade-off decisions made by the DBMS designer/implementor require a very detailed look at the performance aspects of the underlying, physical architecture. This means the generalized performance measurement objective of the DBMS designer/implementor is to obtain a detailed, overall picture of DBMS performance to find those areas not providing an acceptable level of service. The third observer at this level is the DBA, and since the DBA has the responsibility for the overall control of the DBMS and data base(s), the DBA needs an overall picture of DBMS performance to compare with some benchmark level of performance. Additionally, at this level of observation, the DBA must be able to determine the physical I/O activity on the channels and devices, I/O reference patterns, and the distribution of data base files across the mass-storage system. Therefore, at this level of observation, the DBA has the <u>generalized performance measurement objective</u> of determining a concise picture of overall DBMS performance as well as the details of physical I/O activity to see if these performance aspects are within acceptable limits. Failure to meet performance objectives at this level may indicate the DBA must perform a more detailed performance analysis at the other two levels of DBA observation.

System Requirements

The preceding section developed the processes of computer system and DBMS performance evalaution. Data flow diagrams of these processes were presented in Figures II-4 and II-5, respectively. This section concentrates on the Service Workload and Service DBMS Workload processes contained in these diagrams. The goal of this section is to define the set of performance parameters which must be measured by a DBMS performance monitor. The development of this set of parameters is a necessary first step because the measured values of these parameters provides the vehicle for evaluating DBMS performance. Also, the relationship of this set of parameters to the architectural model of a DBMS and the different types of DBMS users is developed.

Computer System Performance Evaluation Parameters. Since the performance monitoring of a DBMS has been defined as a subset of computer system performance monitoring, the logical place to begin is with the parameters commonly used to evaluate the performance of a general purpose computer system. The architectural implementations and monitoring facilities of computer systems tend to have a wide degree of variation; therefore, a complete list of computer system performance parameters is almost impossible to develop (ref. 46:82 and 24:64-66). Consequently, a generalized, partially complete example of computer system performance parameters grouped by their corresponding performance index was developed.

Figure II-7 illustrates how performance parameters are derived. The workload characteristics determine the type and amount of resources the system must allocate to process the users workload. Some examples of workload characteristics are the CPU time and amount of memory requested by a job. Additional examples of workload characteristics are provided in Table B-1 of Appendix B (ref. 46:12-13).

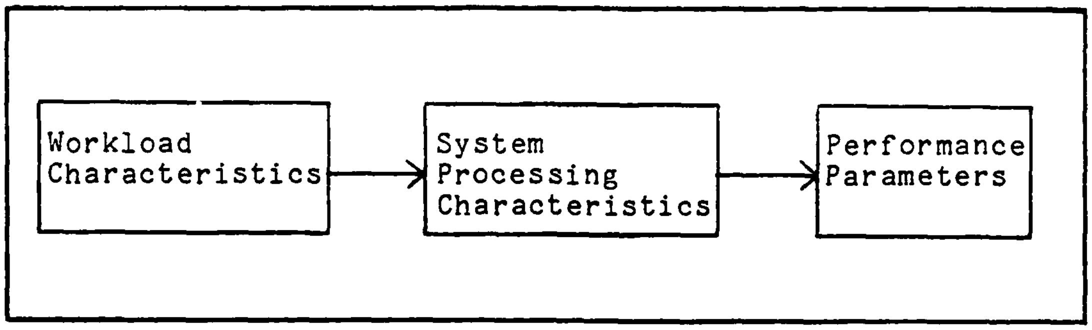


Figure II-7. Derivation of Computer System Performance Parameters

The system processing characteristics determine how the system resources are utilized during the processing of the users workload. Examples of system processing characteristics are throughput, turnaround/response time, and component utilization. Additional examples of system processing characteristics are provided in Table B-2 of Appendix B. Table B-2 was developed from information contained in the following sources: ref. 26:7; 27:; 33:112; 46:16-18; as well as information derived during the course of this study.

The combination of workload and processing characteristics forms the foundation for developing the performance parameters of a given computer system, and the measured values of these performance parameters forms the foundation for evaluating computer system performance. Some examples of computer system performance parameters are shown in Table II-2. A generalized, partially complete set of computer system performance parameters was

TABLE II-2

Examples of Computer System Performance Parameters

Parameter Name	Description
Throughput	Number of jobs executed per unit of time
Turnaround time	Elapsed time between submitting a job or interactive command to a system and receiving the output
Component Reliability	Probability of the component being active and working correctly at any given time
Number of disk Volumes	Number of disk volumes requested by a job
CPU idle	Percentage of time the CPU was idle
CPU busy	Percentage of time the CPU was busy
Page rate	Rate at which pages are retreived from secondary storage
Number of page faults	Number of page faults per unit of time
I/O System idle	Percentage of time the I/O system was idle
I/O System busy	Percentage of time the I/O system was busy
Device idle	Percentage of time a device was idle
Device busy	Percentage of time a device was busy
CPU only	Percentage of time the CPU was the only active component
CPU wait	Percentage of time the CPU spent waiting for I/O to complete
CPU and any I/O	Percentage of time the CPU and any I/O occurred simultaneously
Mean length of system queues	Mean number of units contained in the queue and the percentage of time the queue was this size

developed, and this set of parameters is presented in Table B-3 of Appendix B. The set of performance parameters presented in Table B-3 are categorized and grouped by the performance index scheme of Figure II-2. Table B-3 was developed from information contained in the following sources: ref. 24:Chapter 2; 25:Appendix B; 26:Chapter 1; 27:; 46:10-20; as well as information derived during the course of this study. For Table B-3 to be put into practical use, the source for measuring values for these parameters needs to be defined, and this is covered in the Performance Monitoring Requirements section of this chapter.

The computer system performance parameters describe the way the Service Workload process in Figure II-4 is monitored to produce a set of system effectiveness and a set of system efficiency measures. Figure II-8 contains an expansion of the Service Workload process. The computer workload is input to an Evaluate Workload process which corresponds to the job scheduler of a computer system. In this process, the workload is evaluated to determine its resource requirements, and information about the workload in the form of workload parameters (characteristics) are extracted. For a computer system to execute a user's job, it needs to allocate resources to the job, utilize the resources to execute the job, and return the resources to the system when the job has

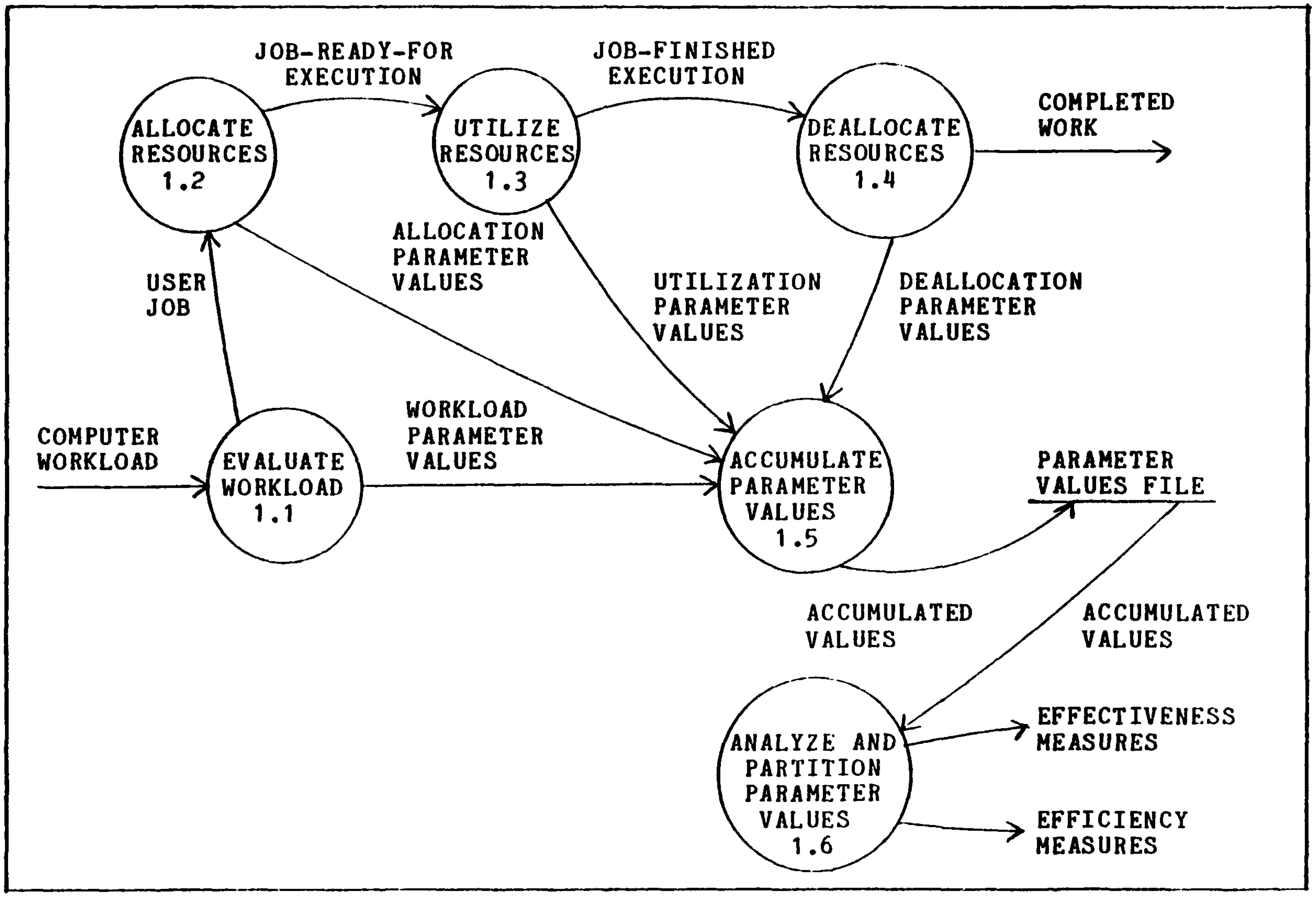


Figure II-8. Service Workload

finished execution. The execution process is shown by bubbles 1.2, 1.3, and 1.4, and each of the processes extracts information about the execution of the job in the form of allocation, utilization, and deallocation parameters. All of the extracted parameters are accumulated within the storage of the system, and at the end of some specified time interval, the accumulated parameters are analyzed and partitioned to produce the effectiveness and efficiency measures which are used to evaluate the performance of the computer system.

Relationship of These Parameters to a DBMS. In Part One of this chapter, the concept of the boundary between system workload and system resources was developed, and three types of computer system resources were defined. Ignoring hardware and human resources, it was shown the software resources of a computer system can be placed on either one or both sides of the boundary depending on an individual's level of observation. An evaluation of the developed set of performance parameters showed the parameters represent information only at the computer system level of observation (i.e. the software resources are measured as a part of the overall system workload). Therefore, if the software resources are to be classified as a system resource, the set of parameters must be modified to include specific information on how software resources are utilized and their impact on the other system resources. For example, a conventional DBMS composed entirely of software modules may be used in a computer system where the CPU utilization is 80 percent; however, the computer system performance parameters developed to this point contain no way to determine how much of the 80 percent CPU utilization was dedicated to supporting the DBMS.

The example shows the parameters developed so far do not meet the performance monitoring needs of all the DBMS users defined in Figure II-6. Therefore, there must be some performance parameters unique to the software resources of a computer system as well as some performance parameters unique to a particular type of DBMS. An evaluation of how the type of DBMS effects the development of a set of performance parameters for DBMS performance monitoring is given below.

Conventional DBMS. An example of how a conventional (Single Machine) DBMS may be implemented on a general purpose computer system is shown in Figure II-9. Since this type of DBMS is composed entirely of software modules, a conventional DBMS is a software resource of the system, and the developed list of performance parameters needs to be modified to include information on how the

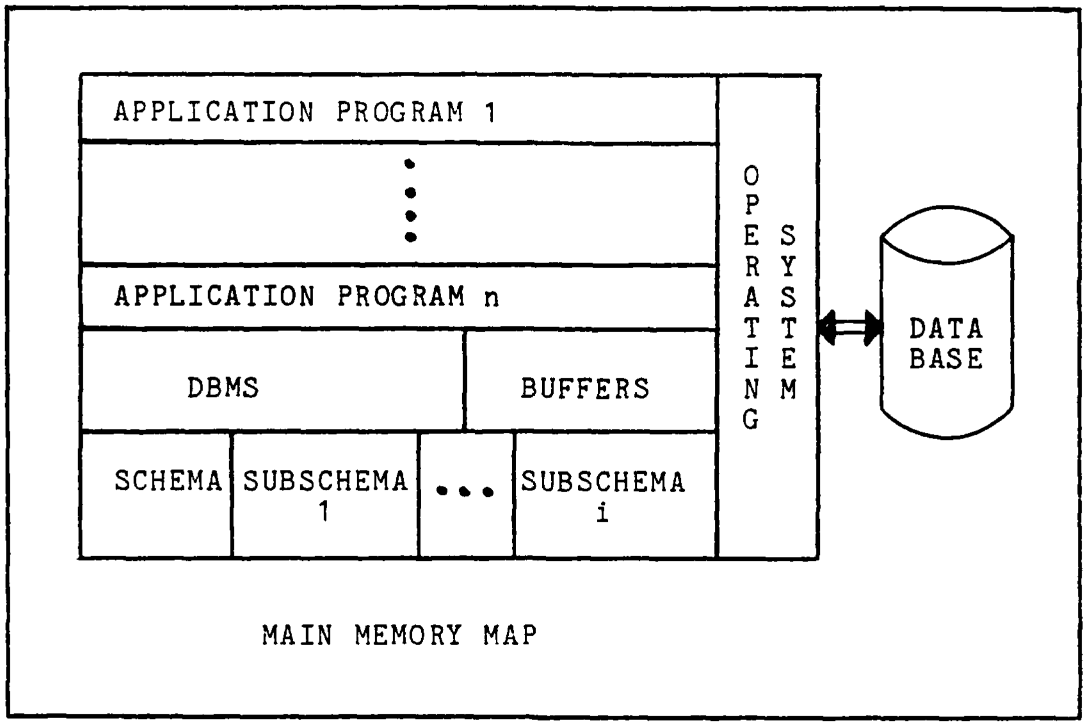


Figure II-9. Conventional DBMS Implementation

DBMS is utilized as well as how it impacts on the other resources of the computer system.

<u>Data Base Computer</u>. An example of how a data base computer may be implemented is shown in Figure II-10. This figure shows three separate aspects must be monitored. First, the DBMS applications programs, interface software, and communications hardware/software must be monitored on the host computer. Second, the data base computer and its corresponding interface software and communications hardware/software must be monitored, and lastly, the use of these two computers as an integrated

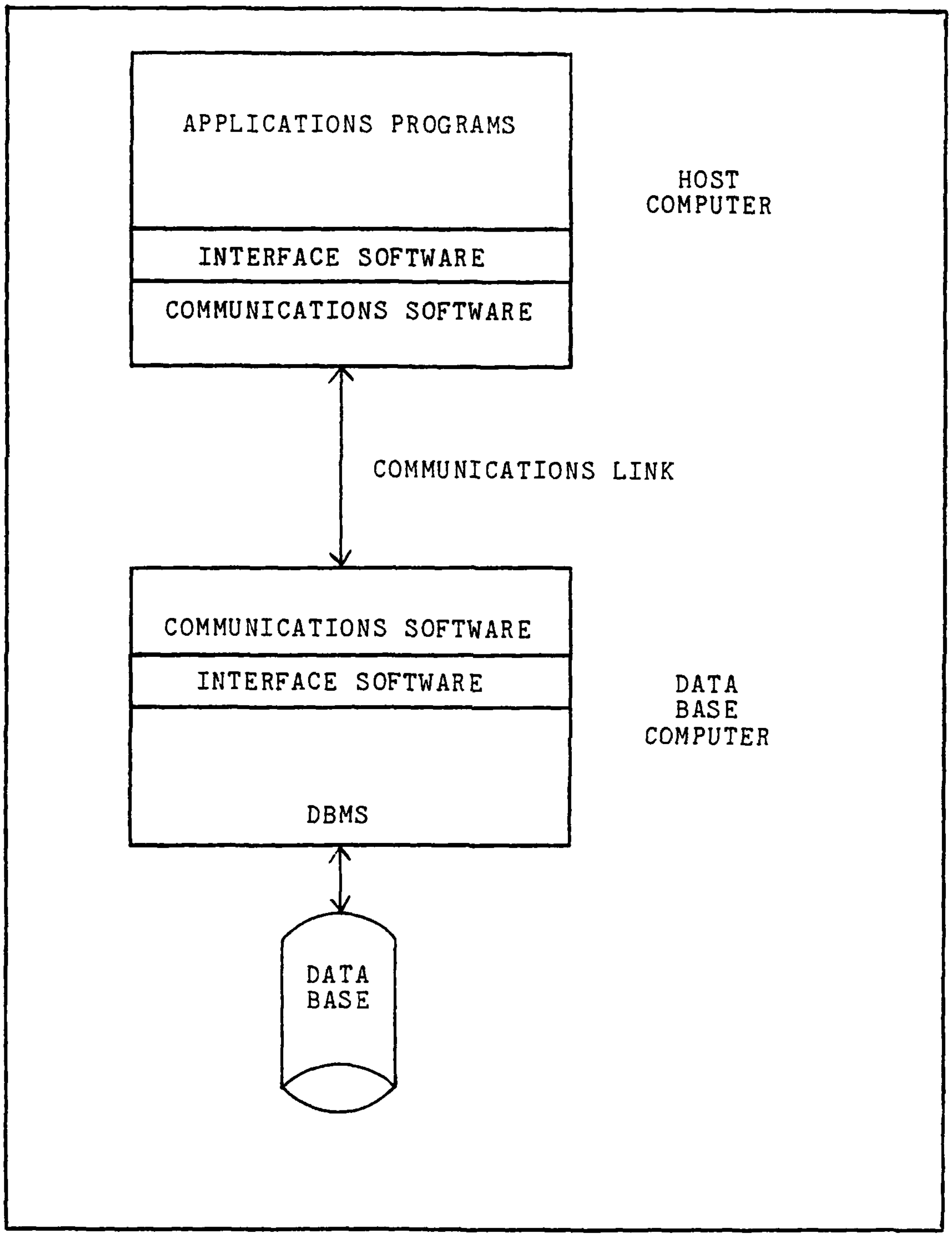


Figure II-10. Data Base Computer Implementation

system must be monitored to provide a complete picture of DBMS performance.

By adding parameters to measure the interface software and the communications hardware/software as a part of the system workload, the developed set of computer system performance parameters could be used to monitor the performance aspects of the host computer. Monitoring the data base computer may not be as easy. The data base computer could be a general purpose computer using a conventional DBMS, and in this case, the same parameters used to monitor the performance of a conventional DBMS could be used. However, the data base computer may contain specialized hardware to perform all or some of the DBMS functions. In this case, many of the parameters used to monitor the performance of a conventional DBMS would apply, but the data base computer's specialized hardware resouces requires the definition of specialized performance parameters. Therefore, monitoring the performance of a data base computer involves more than just modifying the developed set of performance parameters. Since the scope of this study was limited to monitoring the performance of conventional DBMSs, the problem of monitoring data base computers implemented with specialized hardware was not further developed.

The overall picture of DBMS performance is obtained by combining performance parameters from both the host computer and data base computer.

Distributed Data Base System. An example of how a distributed data base system may be implemented is shown in Figure II-11. This is the hardest system to monitor, especially if a picture of overall performance is desired. In this system, four separate aspects must be monitored. First, the performance of each DBMS at the nodes of the network needs to be monitored where the nodes could be conventional DBMSs or data base computers. Second, the performance of the network needs to be monitored (a significant performance monitoring effort in itself ref. 47:Chapter 5). Third, the integrity of the data in the entire network of data bases needs to be monitored, and lastly, the combination of these three aspects must be monitored to develop a complete picture of DBMS performance. In this study, only the first aspect of conducting a performance monitoring effort on a distributed data base system will be investigated, and the investigation of this aspect is limited only to those nodes using a conventional DBMS or a backend data base computer implemented with a general purpose computer and conventional DBMS.

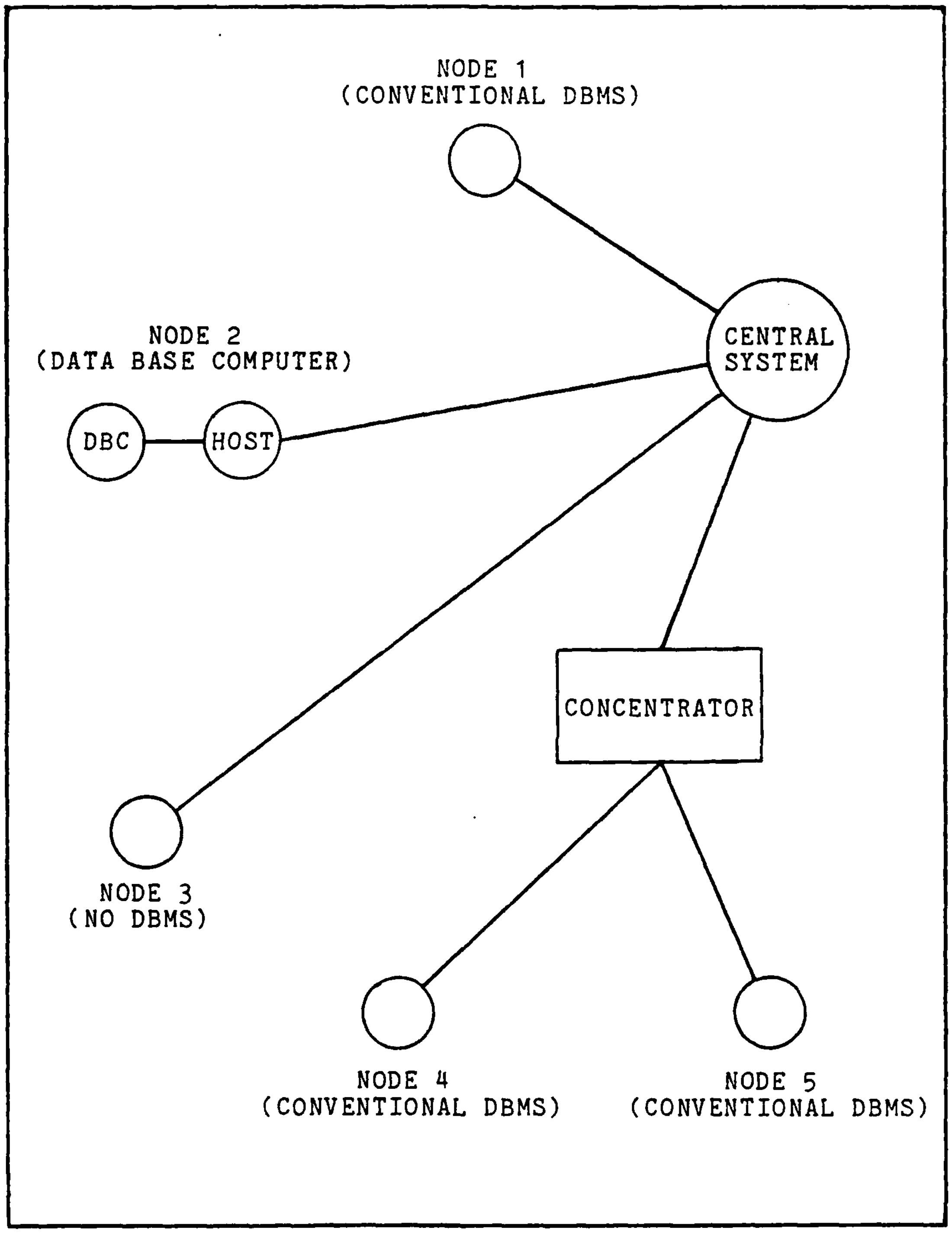


Figure II-11. Distributed Data Base Implementation

Performance Parameters Unique to a DBMS. Figure II-12 illustrates how performance parameters for a DBMS are derived. In a conventional DBMS, the DBMS workload characteristics determine two things: the DBMS software modules that must be used as a system resource and the type and amount of other resources the computer system must allocate to support the DBMS software modules. In fact, data base workload plays a large part in determining the performance of a DBMS, and it is characterized by the list of data objects required to satisfy a request for information contained in a data base(s) (ref. 52:2,12). Some examples of DBMS workload characteristics are the CPU time requested by a DBMS task and the rate at which requests for information are submitted to the DBMS. Additional examples of DBMS workload characteristics are provided in Table B-4 of Appendix B.

Data base workload is represented by the language used to communicate with the DBMS. This language is referred to as the data sublanguage (DSL), and it is some subset of the total computer system language(s) used to access data base objects and specify data base operations (ref. 11:19-21). The data sublanguage could be an interactive query language, statements embedded within an applications program, or specialized statements interpreted by an applications program. This study makes

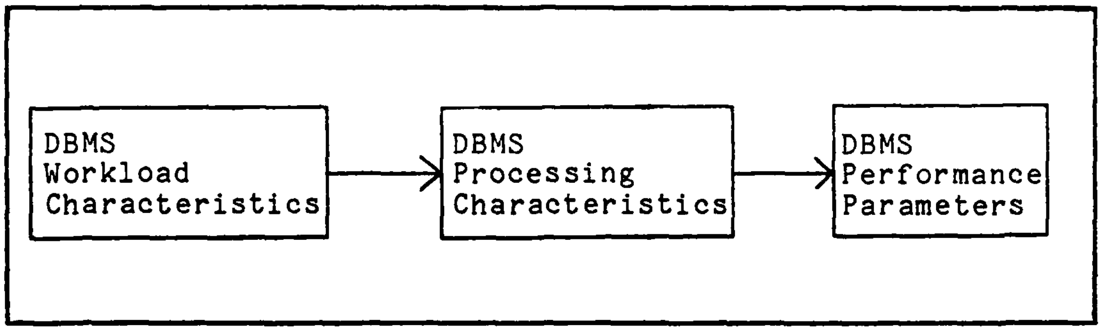

DBMS Workload Characteristics
DBMS Processing Characteristics
DBMS Performance Parameters

Figure II-12. Derivation of DBMS Performance Parameters

no distinction on data sublanguages and treats them all as a generalized way to access and manipulate data in the data base. Any given data sublanguage is a combination of two languages. First, there is a data definition language (DDL) used to describe data base objects, and second, there is a data manipulation language (DML) used to manipulate and process data base objects. The data manipulation language contains statements allowing DBMS users to retrieve objects, create objects, update objects, load a data base, etc.. Therefore, they are the biggest contributor to the DBMS workload. On the other hand, statements in the data definition language usually serve only to describe a particular user's view of the data base and contribute only a small part to DBMS workload. Based on this premise, this study focuses on DML statements and their impact on the system. Four general types of DML statements can be defined and these are given below (ref. 7:49-52):

1. DML retrieval statements - This type of DML statement requires that data base objects or information be moved from a lower level to a higher one. (The lower level would be the mass-storage files of the data base which is the lowest level of the storage hierarchy, and the higher level would be main memory which is the highest level of the storage hierarchy).

2. DML storage statements - This type of DML statement requires that data base objects or information be moved from a higher level to a lower level.

3. DML control statements - This type of DML statement does not require data movement. Rather, it prepares the data base for data manipulation. Examples of this type of operation would be statements to OPEN and CLOSE the data base for processing.

4. Special purpose DML statements - A special purpose DML statement is any DML statement which cannot be classified as any of the other three types of DML statements.

DBMS processing characteristics determine the way DBMS software modules and other system resources are utilized during the processing of the DBMS workload. Examples of DBMS processing [illegible]racteristics are DBMS throughput, the num[illegible] I/ [illegible]per[illegible]ions per DML statement, and the transfer time [illegible] [illegible]ase objects between storage hierarchies. Additional examples of DBMS processing characteristics are provided in Table B-5 of Appendix B.

The combination of workload and processing characteristics forms the foundation for developing the performance parameters of a conventional DBMS, and the measured values of these performance parameters form the foundation for evaluating the performance of a conventional DBMS. Some examples of DBMS performance parameters are shown in Table II-3. A generalized, partially complete set of DBMS performance parameters, grouped by their corresponding performance index, was developed, and this set of parameters is presented in Table B-6 of Appendix B. The data presented in Tables B-4, B-5, and B-6 of Appendix B was developed from information contained in the following sources: ref. 2:315-321; 7:23, 47,49-55,113-124; 13:21,89-94; 28:Chapter 4; 29:4-7; 31:252; 39:330; 42:7; 48:23,26-29; and 52:75,103-105; as well as information derived during the course of this study. For Table B-6 to be put into practical use, the source for measuring values for these parameters needs to be defined, and this issue is covered in the Performance Monitoring Requirements section of this chapter.

The set of DBMS performance parameters developed in this study describe how the Service DBMS Workload process in Figure II-5 is monitored to produce a

TABLE II-3

Examples of DBMS Performance Parameters

Parameter Name	Description
DBMS throughput by type of DML statement	Number of DML statements executed per of time broken down into the four categories of DML statements
Turnaround time	Elapsed time (in batch mode of operation) between submitting a DBMS task and receiving the output
Response time	Turnaround time for a DML statement or set of DML statements in an interactive mode of operation
Number of data bases	Number of data bases requested by a DBMS task
DBMS CPU utilization	CPU usage by the DBMS / total CPU usage
Mean CPU time per DML statement	Mean CPU time to complete a single DML statement
DBMS memory utilization	Memory usage by DBMS tasks / total available memory
DBMS I/O utilization	I/O usage by the DBMS / total I/O usage
Number of data base objects accessed	Total number of data base objects retreived or stored from/into a data base(s)
DBMS device utilization	Device usage by the DBMS / total device usage
Mean length of DBMS queues	Mean number of units contained in the queue and the percentage of time the queue was this size
DBMS functions used	Number of times specific DBMS modules were used during a DBMS task

set of DBMS effectiveness and a set of DBMS efficiency measures. Figure II-13 contains an expansion of the Service DBMS Workload process. The DBMS workload is input to an Evaluate Workload process which corresponds to the task scheduler or control module of the DBMS. In this process the DBMS workload, in the form of DML statements, is evaluated to determine the resource requirements in terms of DBMS function modules and data base files, and information about the DBMS workload, in the form of DBMS workload parameters (characteristics), is extracted. For the DBMS to execute the DML statement, it needs to allocate resources, utilize the resources, and return the resources when the DML statement has completed its execution. The DML statement execution process is shown by bubbles 1.2, 1.3, and 1.4, and each of the individual processes extracts information about the execution of the DML statement in the form of allocation, utilization, and deallocation parameters. All of the extracted parameters are accumulated within the storage of the host computer system, and at the end of some specified time interval, the accumulated parameters are analyzed and partitioned to produce the effectiveness and efficiency measures used to evaluate the performance of the DBMS.

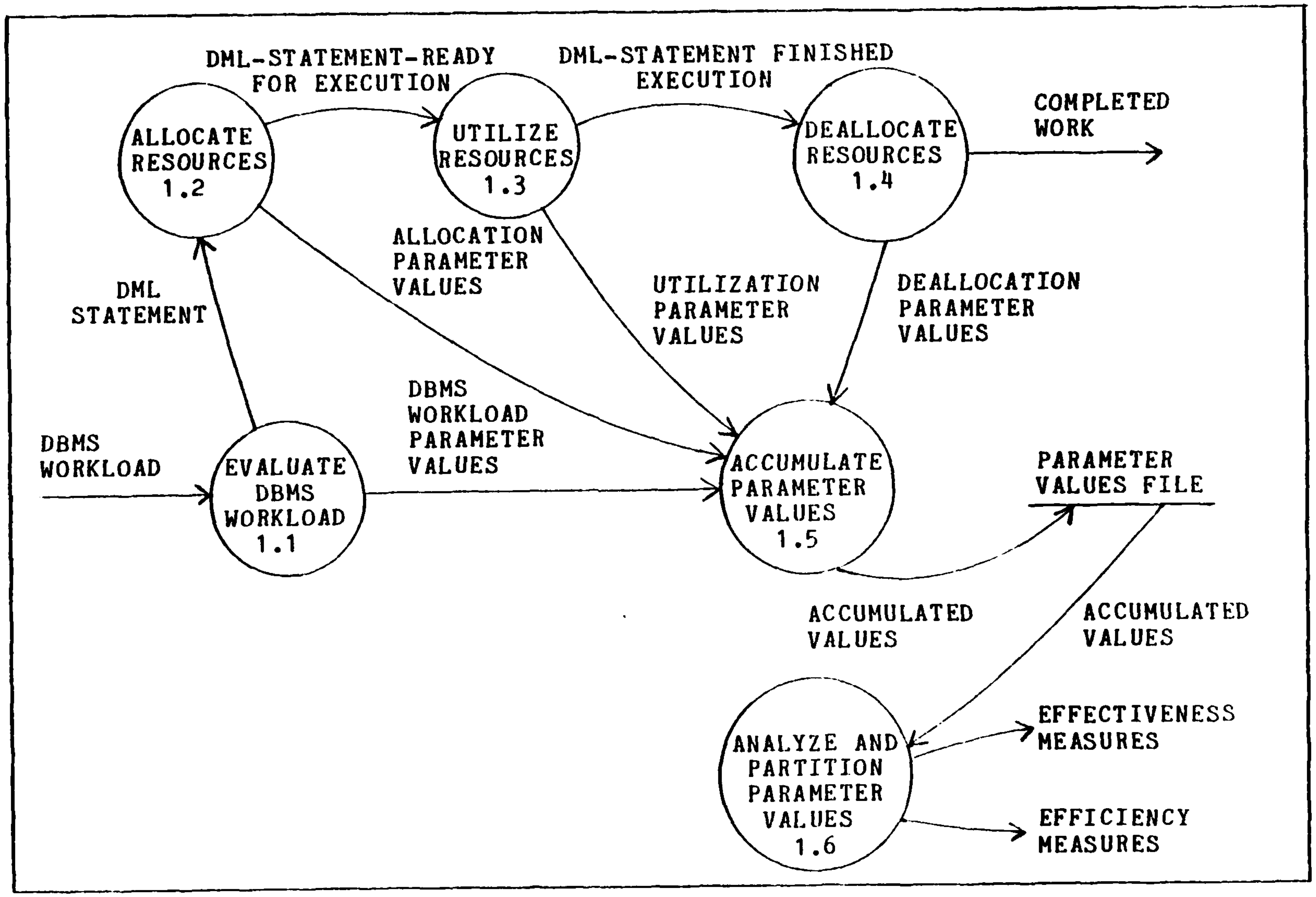

DML-STATEMENT-READY FOR EXECUTION
DML-STATEMENT FINISHED EXECUTION
ALLOCATE RESOURCES 1.2
UTILIZE RESOURCES 1.3
DEALLOCATE RESOURCES 1.4
COMPLETED WORK
ALLOCATION PARAMETER VALUES
DML STATEMENT
UTILIZATION PARAMETER VALUES
DEALLOCATION PARAMETER VALUES
DBMS WORKLOAD
EVALUATE DBMS WORKLOAD 1.1
DBMS WORKLOAD PARAMETER VALUES
ACCUMULATE PARAMETER VALUES 1.5
PARAMETER VALUES FILE
ACCUMULATED VALUES
ACCUMULATED VALUES
ANALYZE AND PARTITION PARAMETER VALUES 1.6
EFFECTIVENESS MEASURES
EFFICIENCY MEASURES

Figure II-13. Service DBMS Workload

Selection of a Set of Performance Parameters for Monitoring a DBMS. The developed sets of performance parameters for computer system and DBMS performance monitoring were used to develop a combined set of performance parameters meeting the performance monitoring needs of the different DBMS users. The new set of performance parameters was developed by combining the parameters from both of the previously defined sets and selecting those parameters meeting the generalized performance measurement objectives and performance monitoring needs of the different DBMS users. Some examples of the combined set of performance parameters are presented in Table II-4. Table C-1 of Appendix C contains the entire set of combined performance parameters, and while this table should not be considered a complete list, it should prove to be comprehensive enough to effectively monitor a conventional DBMS for all the DBMS users.

Relationship to DBMS Architecture and DBMS Users. Table II-5 shows how the example parameters of Table II-4 relate to DBMS architecture and the different types of DBMS users defined in Figure II-6. Table II-5 was developed using the generalized performance measurement objectives of the different types of DBMS users stated earlier.

TABLE II-4

Examples of a Combined Set of Performance Parameters

Parameter Name	Description
DBMS throughput by type of DML statement	Number of DML statements executed per unit of time broken down into the four categories of DML statements
System turnaround time	Elapsed time between submitting a user job or interactive command to the system and receiving the output
DBMS turnaround time	Elapsed time (in batch mode of operation) between submitting a DBMS task and receivning the output
CPU busy	Percentage of time the CPU was busy
DBMS CPU utilization	CPU usage by the DBMS / total CPU usage
Number of page faults (system)	Number of page faults per unit of time attributable to all system processes and user jobs
Number of page faults (DBMS)	Number of page faults per unit of time attributable to the DBMS
I/O busy	Percentage of time the I/O system was busy
DBMS I/O utilization	I/O usage by the DBMS / total I/O usage
Device busy (system)	Percentage of time a device was busy
DBMS device utilization	Device usage by the DBMS / total device usage
CPU and any I/O	Percentage of time the CPU and any I/O occurred simultaneously
CPU and any DBMS I/O	Percentage of time the CPU and any DBMS I/O occurred simultaneously
Mean length of system and DBMS queues	Mean number of units contained in the queues and the percentage of time the queues were this size

TABLE II-5

Relationship of Performance Parameters to the DBMS User

	External - Conceptual Interface (above)	External - Conceptual Interface (below)	Internal Level - Access Method Interface (above)	Internal Level - Access Method Interface (below)		
Performance Parameter	Software Engineer	DBA	DBA	System Manager	DBMS Designer	DBA
DBMS throughput		X		X	X	X
CPU busy				X	X	X
DBMS CPU utilization		X		X	X	X
Number of page faults (system)			X	X	X	X
Number of page faults (DBMS)	X	X	X	X	X	X
I/O busy			X	X	X	X
DBMS I/O utilization		X	X	X	X	X
Device busy			X	X	X	X
DBMS device utilization			X	X	X	X

Using these objectives, the entire set of combined performance parameters contained in Table C-1 was related to the DBMS architectural model and the different types of DBMS users, and these relationships are presented in Table C-2 of Appendix C. Since Table C-2 was developed using generalized objectives, individual DBMS users may need to add or subtract parameters to meet specific needs. To add or subtract parameters, an individual DBMS user simply needs to define specific performance measurement objectives and then select the appropriate parameters necessary to accomplish those specific objectives. For example, a software engineer may want to measure only the CPU and I/O times of a specific DBMS task broken down by the specific DML statements contained in the DBMS task. By doing this the software engineer can locate those DML statements using the most CPU and I/O time and formulate certain design decisions and alternatives that provide better performance.

In developing the tables relating DBMS users to the combined set of performance parameters, the information or management level user of the DBMS was not evaluated. Since these users typically have a non-technical background and are only interested in "visible" response times, they probably have no interest in monitoring different aspects of DBMS performance. Their displeasure with DBMS performance would be passed on to the technical staff in their organization who would use the data provided by the performance parameters to investigate and hopefully correct their dissatisfaction.

<u>Level of Performance Measure Provided by these Parameters</u>. The level of performance measure desired from a DBMS performance monitor is stated by Atre "In a data base environment, continuous, long-term measurements are necessary to understand the service provided by a DBMS. The performance monitor should provide a detailed as well as a bottom line picture of the demand on the data base service, the service provided, the resource consumption in the delivery of that service, and any resource overcommitments. The monitor should be inexpensive to operate and should create concise reports on a few pages" (ref. 2:315). A DBMS performance monitor that measures values for all the parameters contained in the combined set of performance parameters developed in this study meets the criteria called for by Atre; however, this would be monitor "overkill" in many cases. A DBMS performance monitor that takes into consideration a user's performance monitoring objectives should provide a better overall level of performance measurement. Therefore, the results of relating the performance parameters to different types of DBMS users should be the basis for the design or selection of a DBMS performance monitor because

it provides the level of measurement required of a generalized or specific DBMS performance monitoring effort.

Performance Monitoring Requirements

Defining the performance parameters of interest is only the first step in evaluating the Service Workload and Service DBMS Workload processes shown in Figures II-8 and II-13. The next step is to determine how values for the parameters are measured. This section of the study addresses this issue. The section covers: the possible sources of values for performance parameters, the selection/design of a performance monitor, how a user interfaces with a performance monitor, and how data collected by a performance monitor should be analyzed and presented to the monitor user.

Sources for Performance Parameter Values. In this study, general purpose performance monitoring tools such as hardware and software monitors, accounting packages, etc. were evaluated as potential sources of values for the performance parameters used to characterize DBMS performance. However, the ger›ral purpose tools could not measure all the performance parameters used for DBMS performance evaluation. To measure values for the remaining performance parameters, a specialized tool needed to be designed and developed. This new tool is called a DBMS instrumentation program/utility.

Table II-6 shows some of the possible sources of values for the example performance parameters of Table II-4. The sources indicated in the table are: software monitors (Soft Mon), hardware monitors (Hard Mon), job accounting packages (Accnt Data), DBMS instrumentation program (Instr Prog), DBMS log facilities (DBMS Log), calculations based on one or more other performance parameter values (Calc), and error log facilities (Error Log). In the table, values for parameters marked with a 'X' are directly measureable by the indicated source. Whenever a value must or can be calculated, an 'X' is placed in the Calc. column and the possible source(s) of the values used in the calculation are marked with an 'S'. Table C-3 of Appendix C shows the possible sources of values for the entire set of combined performance parameters developed in this study. (ref. Table C-1).

Since Tables II-6 and C-3 contain possible sources, performance monitors already available or being designed for an existing system must be individually evaluated in terms of the set of performance parameters they can measure. The major criteria for performing this evaluation are the monitor's domain, accuracy, and artifact (ref. 24:29-31 and 46:80-93).

TABLE II-6

Example Sources for Performance Parameters

Performance Parameter	Soft Mon	Hard Mon	Accnt Data	Instr Prog	DBMS Log	Calc.	Error Log
DBMS throughput				X	X		
CPU busy	X	X	S			X	
DBMS CPU utilization	X		S		S	X	
Number of page faults (system)	X	X	S			X	
Number of page faults (DBMS)	X		S	S	S	X	
I/O busy	X	X	S			X	
DBMS I/O utilization	X		S	S	S	X	
Device Busy	X	X					
DBMS device utilization	X		S			X	

Relationship of the Source of Performance Parameter Values to the Domain, Accuracy, and Artifact of the Performance Monitor. It was previously mentioned that even though the examples of performance parameters presented in Tables II-4 and C-1 are a comprehensive set, an individual DBMS user may want to measure only a subset of this overall set, and some guidelines for selecting a subset of performance parameters based on a individual DBMS user's performance measurement objectives were shown in Tables II-5 and C-2. Unfortunately, there is another limiting factor in the selection of a subset of performance parameters to be used for a particular DBMS performance monitoring effort. This additional limiting factor is the domain of the performance monitor used to accumulate the performance measurement data. Therefore, the subset of performance parameters may not be entirely measureable by a single performance monitor, and this leads to a trade-off situation in which the subset of performance parameters may be further modified to fit the domain of the performance monitor. Alternatively, more than one performance monitor, each with a different domain, can be used to obtain measurement values for the entire subset of performance parameters. The particular trade-off decision depends on the established performance monitoring objectives and the budget allowed for the performance monitoring effort. The budget must be considered since the use of additional performance monitors will increase both the system overhead costs in terms of monitor artifact and the cost of purchasing, leasing, or designing and developing the additional monitors.

Selection/Design of a Performance Monitor. The performance tools listed in Tables II-6, C-3, and C-4 do not meet many of the performance monitoring objectives of DBMS users when used singularly, but when some combination of the tools are used in parallel, many or all of the performance monitoring objectives reflected by a set of performance parameters for monitoring DBMS performance can be achieved. If it is possible to use existing performance monitoring tools, the overall design problem of a DBMS performance monitor can be drastically reduced.

To select existing performance monitoring tools for use in the design and development of a DBMS performance monitor, the three step procedure presented below was followed.

1. Well defined operational objectives for the DBMS performance monitor were established.

2. Based on the performance measurement objectives and the guidelines of Table C-2, the set of performance parameters that provides the level of

performance measurement required of the different types of DBMS users was selected.

3. The performance monitoring tools already available on the system were evaluated as well as any new tools available from vendors. The criteria for this evaluation was how well the tools measured the selected set of performance parameters along with the accuracy and artifact of the tools. Tables C-3 and C-4 proved to be very useful for performing this evaluation. Based on the results of this study, the tools listed in Tables II-6, C-3 and C-4 should be evaluated in the following order, and in each case, the measurable values must be evaluated to determine what calculations can be made to extend the set of possible measurable values.

a. Job Accounting Package.
b. DBMS Log Facility.
c. Error Log Facility.
d. Software Monitor.
e. Hardware Monitor.

The order of presentation of these tools is based on several factors. The first factor is the level of performance measurement they provide. The tools at the beginning of the list provide a high level view of system performance without producing much monitor artifact (system interference). If a lower level, more detailed view of system performance is required, the tools at the bottom of the list provide the detailed measurements to support this type of study although at the cost of increased monitor artifact. The second factor is their ease of use with the tools at the beginning of the list being the easiest to use. The last factor is the availability of the tools. Based on a survey of the Datapro Manuals (ref. 10:), most computer systems have some type of job accounting capability and most DBMSs are delivered with some type of DBMS log facility. On the other hand, software and/or hardware monitors for a computer system are usually purchased separately.

Unfortunately, these tools did not provide specific DBMS measurements such as the performance parameters pertaining to individual DML statements. If this type of measurement data is absolutely required to achieve the performance monitoring objectives, the only recourse may be to use a DBMS instrumentation program/utility type of performance monitor. These types of monitors are typically not available from vendors and must be designed and implemented in-house or by contract. In fact, it may be necessary to add instrumentation into the actual software modules of the DBMS. Doing this requires a great deal of knowledge about the software implementation of the DBMS, and it entails a high level of risk because new errors could be introduced into the DBMS. Additionally, it will become increasingly difficult to keep up with updates to the DBMS. Therefore, unless the original vendor of the DBMS is going to provide the instrumentation, this avenue is not recommended.

Alternatively, a set of software modules can be designed and implemented to be used to instrument user programs that contain embedded DML statements. The user programs can call the software modules before and after the DML statements are processed by the DBMS, and in this way, the DBMS specific types of performance data on individual DML statements can be obtained. Again, a great deal of system knowledge may be required to design and implement this set of modules and a certain amount of risk is still present. In some cases, such as the VAX 11/780, the operating system provides utilities for just this purpose, and the use of these utilities help to simplify the task.

The procedure outlined above should provide a set of existing tools for most computers and conventional DBMS, and this set of existing tools should provide sources for measuring values for all or the majority of the desired performance parameters. If it does not, it is still a useful procedure to follow because it defines the measurement requirements of the performance monitor to be designed and developed.

Having defined the requirements for the selection/design of the DBMS performance monitor, two more issues still need to be addressed. These are the issue of a user interface to the DBMS performance monitor and the issue of how to analyze and present the data recorded by the DBMS performance monitor. Each of these issues are addressed in detail in the following two sections.

Definition of the User Interface. The user interface for the DBMS performance monitor needs to provide a "user friendly" method for DBMS users to accomplish their performance measurement objectives. First, the users must be able to select a set of performance parameters, and second, the user must be able to activate the performance tool or combination of performance tools to actually measure the system and record the performance parameter values in a specified data file. Third, the user must be able to specify what type of mathematical analysis needs to be performed on the measured data, and lastly, the users need to be able to review the measurement data after it has been recorded and analyzed. Unfortunately, this user interface will not be able to activate any free-standing hardware monitors that must be used; however, it can be designed to provide information on how the hardware monitor must be connected to the system, initialized, and started.

Human factors considerations are becoming an important aspect of software engineering (ref. 3:41). A user interface for any type of system can not ignore these considerations and expect the user to be happy with the product. Literature on this subject ranges from formal techniques using state diagrams (ref. 32:), to common sense design principles such as provide feedback, be consistent, etc. (ref. 44:), to formal studies on the human error process (ref. 38:).

Personal experience in this area has shown state diagrams to be an excellent tool for designing user interfaces as well as some types of complex programs. During the design stage, no attempt is made to specifically define user error procedures, user prompts, or user command languages. Only general notations are made and after the first-cut design has been completed, a set of conventions for the user interface are defined and applied to the design. This approach makes the task of implementing a consistent user interface much easier, and it quickly shows the set of common routines that must be developed for the user interface.

One last item to address is matching the interface to the user's skill level. Skilled users shouldn't be bored with tedious entries, and at the same time, don't abbreviate so much the less skilled user is unable to use the program. If there is a varied skill level in the user population, this should be taken into consideration and different modes of operation such as verbose, normal, and terse (ref. 45:171) can be developed. For the problem of DBMS performance monitoring, all of the users of the DBMS performance monitor should have a good technical background. Therefore, one clear, concise mode of operation should be all that is required.

Analysis and Presentation of the Performance Measurements. Two types of analysis must be performed on the measured values for the performance parameters. First, some of the necessary values must be calculated from the other values, and the use of a mathematical-statistical package such as SPSS (Statistical Package for the Social Sciences) should be considered. Also, many performance tools have data analysis capabilities built into them. Typical types of calculations will include mean values, variances, and information on the underlying sampling distrubution such as minimum and maximum values. The statistical package can also be used to generate histograms, scattergrams, regression models, etc.. Histograms and scattergrams are useful for presenting graphical displays of performance parameter values. For example, a histogram of the CPU time required by each job processed during the measurement interval will quickly show the distribution of CPU time over all the jobs in the measurement interval. Regression models are useful for modelling performance parameters such as turnaround time. For example, the regression model may show the major part of turnaround time is spent in the input queue waiting for the necessary resources to be allocated. Based on this, the input queueing process is concentrated on to determine why user jobs are spenaing the majority of their time in the input queue.

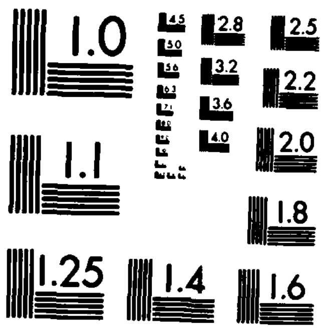
The second type of analysis is taking the performance measurements and using them to formulate or justify performance hypotheses about the computer system and the DBMS. This aspect of analyzing the performance measurements is covered in the Methodology/Procedures section of this chapter.

Presenting the measured values for the performance parameters to the user of the DBMS performance monitor should follow the advice of Atre who states the monitor "should create concise reports on a few pages". In following this advice, measured values for the combined set of performance parameters presented in Table C-1 will be presented in basically the same way they are presented in the table - categorized by performance index. Therefore, a report organized as follows should prove to be satisfactory for this and for other studies.

1. Page One - Page one will contain all

AD-A149 950 DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM PERFORMANCE MONITOR VOLUME 1(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. P D BAILOR 2/3

UNCLASSIFIED DEC 83 AFIT/GCS/EE/83D-2-VOL-1 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

the effectiveness measures grouped by their corresponding category of Productivity, Responsiveness, Integrity, and Security.

2. Page Two - Page two will contain the first and last categories of the efficiency measures which are the allocation and deallocation parameters.

3. Page Three and Four - These pages will contain the utilization parameters grouped by the resource type of CPU, Memory, I/O, Channels, etc..

4. Additional Pages - Additional pages can be added to this report, and these pages will consist of statistical analysis items such as histograms, regression models, hypothesis tests, etc.

<u>Relationship of the Performance Measurements to Controlling the DBMS Environment</u>. One more aspect of measuring values for the performance parameters needs to be investigated, and this aspect is what to do with the values for the performance parameters after they have been measured, mathematically analyzed, and presented to the user of the monitor. In other words, how can the measurement data be used to improve or control DBMS performance and help meet the needs of the DBMS users. Obviously, some aspects of the computer system and/or DBMS need to be changed or modified to realize a performance improvement, and there are several different ways to

change/modify the computer system or DBMS. Four example changes/modifications are given below (ref. 46:20):

1. Adjust the system and DBMS control parameters (i.e. modify the parameters used to generate the operating system and the DBMS).

2. Change or modify resource management policies.

3. Distribute the load among system components to balance resource utilization (e.g. changes in the assignment of peripheral devices to channels or the assignment of files to physical storage devices, changes in the distribution of software components in the system memory hierarchy, etc.).

4. Replace or modify system components.

The examples appear to be straightforward solutions to performance problems, but actually solving the problems is not as easy as it sounds. The difficult part is determining from the measurement data what specific aspects of the computer system and/or DBMS to change or modify to gain the performance improvement. The solution to this problem is non-trivial, especially if some type of structured methodology or procedure is not adhered to (ref. 5:). Therefore, an examination of this problem is presented next.

Methodology/Procedures for Conducting a DBMS Performance Study

The proper development of a methodology for conducting an overall DBMS performance study is important. It provides the individuals conducting a DBMS performance study with procedures for measuring DBMS performance and procedures for using the measurements to make changes to the host computer system and DBMS to increase overall performance. The basis for the material developed in this section comes from the report by Bell, et. al. (ref. 5:). The approach to the material in this section is the same approach used in the preceding sections. First, the material is developed for computer system performance evaluation in general, and then the scope is narrowed by treating DBMS performance evaluation as a subset of computer system performance evaluation. Additionally, the material in this section is supported by the further development of the Data Flow Diagrams used to specify the performance evaluation process. In particular, bubbles 2 and 3 of Figures II-4 and II-5 are expanded.

To begin, the seven phases of a procedure for conducting a computer system performance monitoring effort contained in the report by Bell, et. al. (ref. 5:) are briefly described. For those readers not familiar with this report, it is a valuable source of information and should be read. A copy of this report may be obtained from the Defense Technical Information Center (DTIC) by requesting report AD 737 317.

1. Understand the System - The purpose of this phase is to learn and understand the details of the system to be analyzed. This should include details of its hardware, software, workload, and the organization of the installation management.

2. Analyze Operations - The purpose of this phase is to analyze the management of the system operations. This analysis will help review the operational objectives of the installation and how the management is currently trying to accomplish these objectives.

3. Formulate Performance Improvement Hypotheses - The purpose of this phase is to formulate specific, performance related hypotheses on possible problems and their possible cures. The hypotheses are based on what was learned from the first two phases and any performance data that may be available. For example, the utilization of an individual peripheral device such as a disk drive may be high while the other drives have a low utilization. This data may lead to the hypothesis that re-locating some of the files from the highly used device onto the other devices may reduce the contention for that device thereby decreasing I/O wait times.

4. Analyze Probable Cost-Effectiveness of Improvement Modifications - The purpose of this phase is to critically analyze the hypotheses formulated in phase 3. Due to budget constraints or unrealistic objectives, the performance improvements predicted by the formulated hypotheses may not be worth the additional investment.

5. Test Specific Hypotheses - The purpose of this phase is to test the validity of the hypotheses from phase 3. If the hypotheses are valid, they can be used to implement modifications; otherwise, they must be reformulated or rejected. It is in this phase that specific types of performance monitors are selected and used to collect performance measurement data.

6. Implement Appropriate Combinations of Modifications - The purpose of this phase is to select a set of modifications based on the valid hypotheses and implement them. The modifications should be selected so they do not unduly affect production requirements, and if more than one modification is made, they should not cancel each other.

7. Test Effectiveness of Modifications - The purpose of this phase is to determine the effects of the modifications to see if performance is now satisfactory or if additional analysis and modifications are required to achieve the operational and performance objectives.

The seven phases are easily placed into the form of a Data Flow Diagram. Phases 1 and 2 are the underlying phases for the Determine System Objectives process shown as bubble 2 in Figure II-4. The breakdown of bubble 2 is shown in Figure II-14, and this is an important diagram because it shows how performance objectives are derived. This study has introduced, defined, and stressed the importance of performance objectives in the earlier sections, and now the last factor, how to actually derive a set of specific performance objectives for a performance monitoring effort has been presented.

Phases 3 through 7 are the underlying phases for the Analyze Performance process shown as bubble 3 in Figure II-4. The breakdown of bubble 3 is shown in Figure II-15, and this is also an important diagram since it shows what to do with performance objectives, effectiveness measures, and efficiency measures in order to study and improve the performance of a computer system. Bubbles 3.3 and 3.5 of Figure II-15 deal with the process of testing performance hypotheses (Phase 5) and testing the modifications made to a computer system (Phase 7). Figures II-16 and II-17 contain the breakdown of these two processes. The salient feature of these diagrams is that specific performance tools are not selected for use until a set of hypotheses tests have been designed. This aspect is important because it eliminates situations of using performance tools to

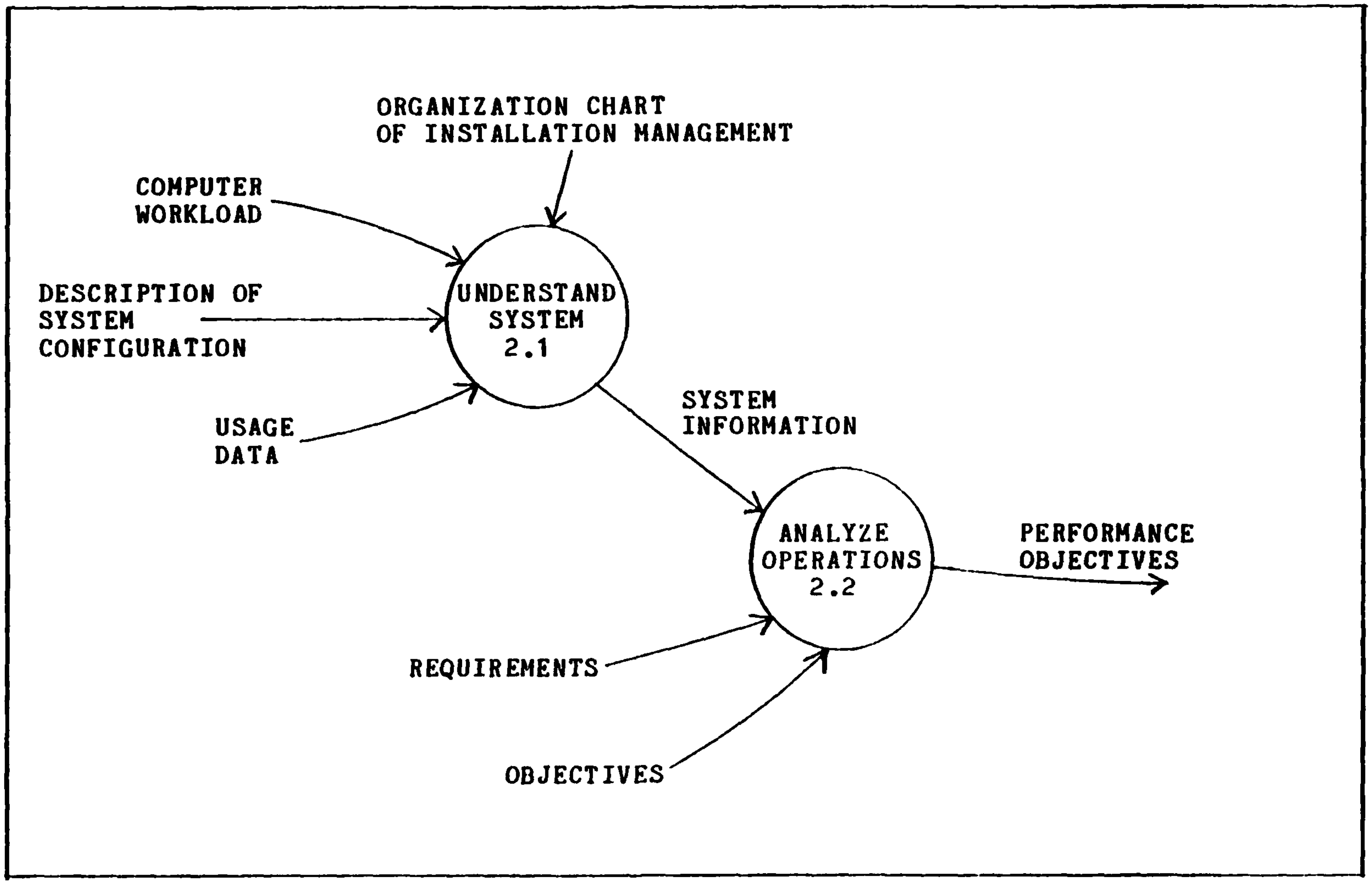


Figure II-14. Determine System Objectives

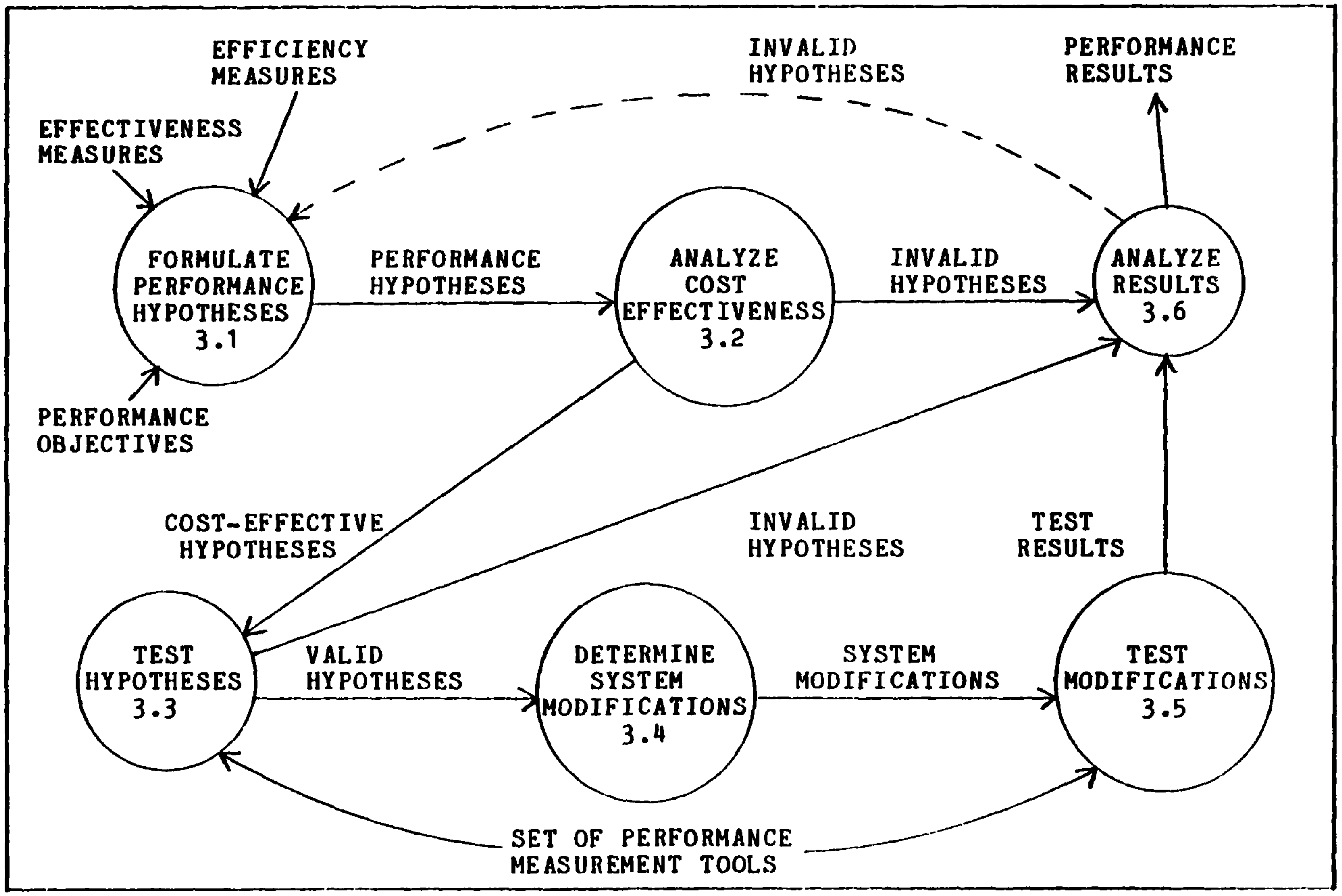


Figure II-15. Analyze Performance

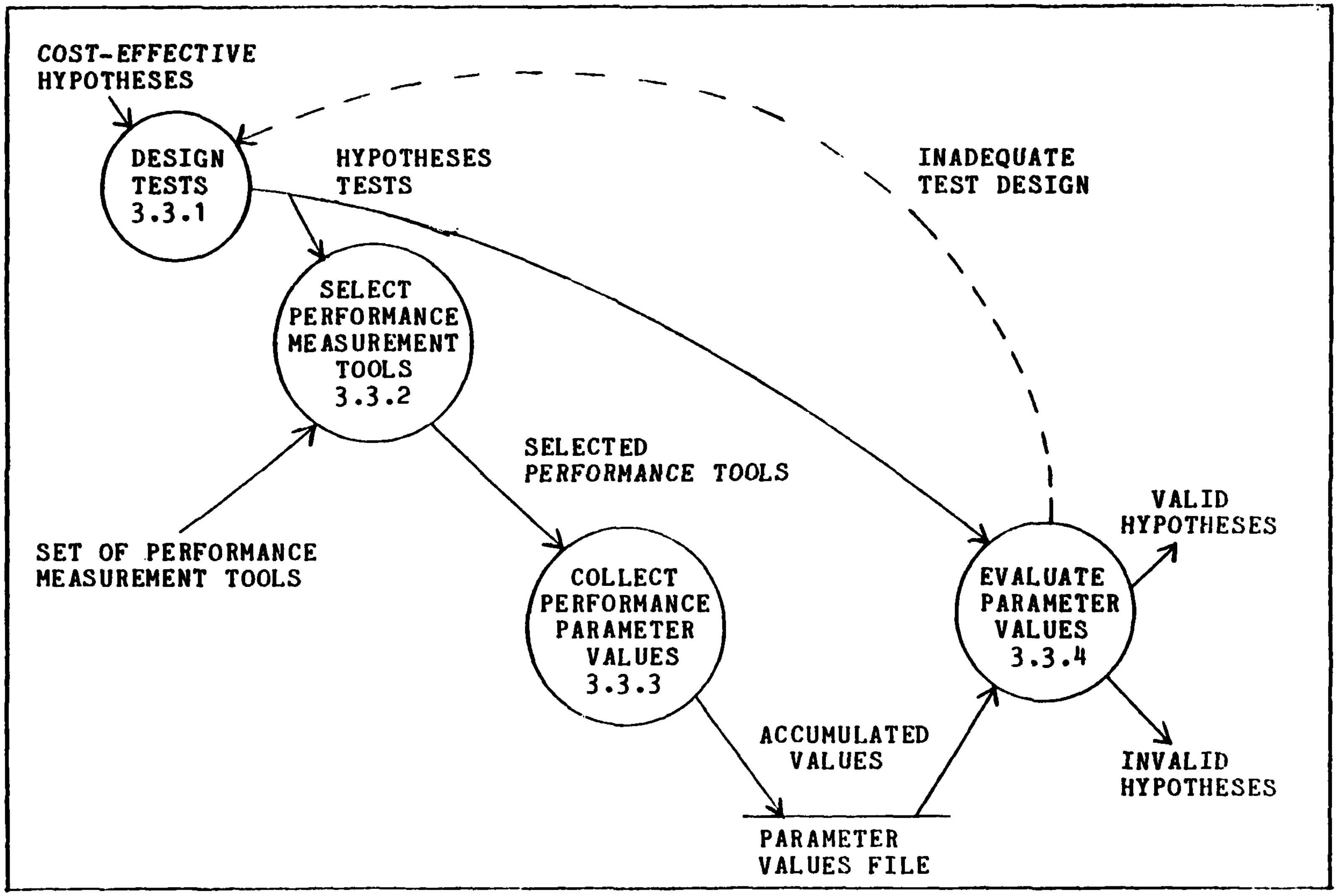

COST-EFFECTIVE HYPOTHESES
DESIGN TESTS 3.3.1
HYPOTHESES TESTS
INADEQUATE TEST DESIGN
SELECT PERFORMANCE MEASUREMENT TOOLS 3.3.2
SET OF PERFORMANCE MEASUREMENT TOOLS
SELECTED PERFORMANCE TOOLS
COLLECT PERFORMANCE PARAMETER VALUES 3.3.3
ACCUMULATED VALUES
PARAMETER VALUES FILE
EVALUATE PARAMETER VALUES 3.3.4
VALID HYPOTHESES
INVALID HYPOTHESES

Figure II-16. Test Hypotheses

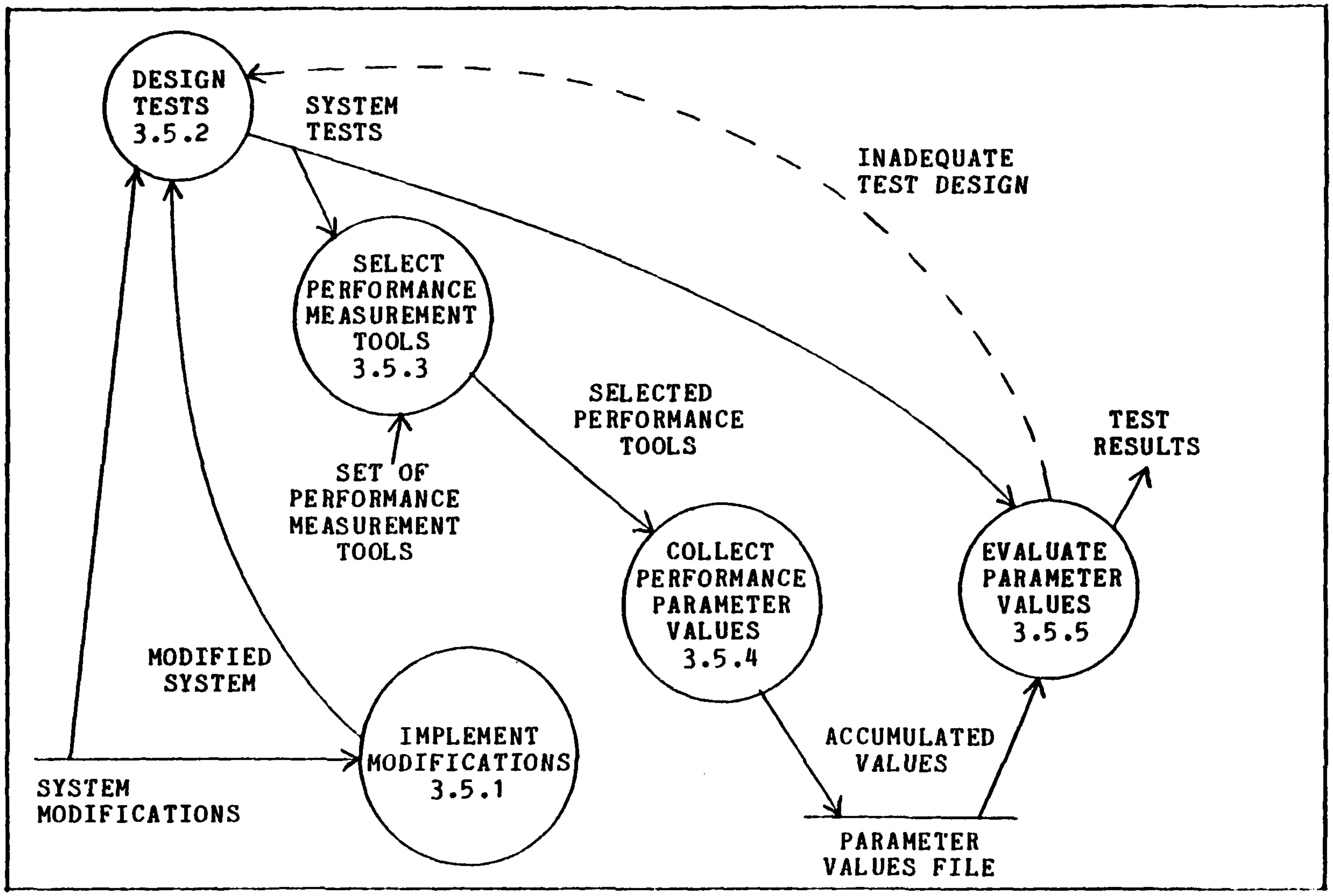

DESIGN TESTS 3.5.2
SYSTEM TESTS
INADEQUATE TEST DESIGN
SELECT PERFORMANCE MEASUREMENT TOOLS 3.5.3
SELECTED PERFORMANCE TOOLS
TEST RESULTS
SET OF PERFORMANCE MEASUREMENT TOOLS
COLLECT PERFORMANCE PARAMETER VALUES 3.5.4
EVALUATE PARAMETER VALUES 3.5.5
MODIFIED SYSTEM
IMPLEMENT MODIFICATIONS 3.5.1
ACCUMULATED VALUES
SYSTEM MODIFICATIONS
PARAMETER VALUES FILE

Figure II-17. Test Modifications

collect performance data and then wondering what to do with the data.

The development of the Data Flow Diagrams for Computer Performance Evaluation is now complete, and these diagrams provide a computer system analyst with the necessary information for conducting a performance evaluation effort on a computer system. All that is left is to narrow the scope of the information to allow it to be applied to a performance evaluation study of a conventional DBMS. The Data Flow Diagram for DBMS Performance Evaluation was shown in Figure II-5. The breakdown of the Determine DBMS Objectives process of bubble 2 is shown in Figure II-18, and the breakdown of the Analyze DBMS Performance process of bubble 3 is shown in Figure II-19. A breakdown of the test hypotheses and test modifications bubbles of Figure II-19 are not shown because they are identical to the corresponding DFDs for computer performance evaluation. These diagrams complete the development of the DBMS Performance Evaluation Data Flow Diagram originally shown in Figure II-5. Therefore, a complete procedure for conducting a DBMS performance study has also been specified.

Functional Requirements

The major questions in any performance evaluation effort are what data to collect, when and how much data to

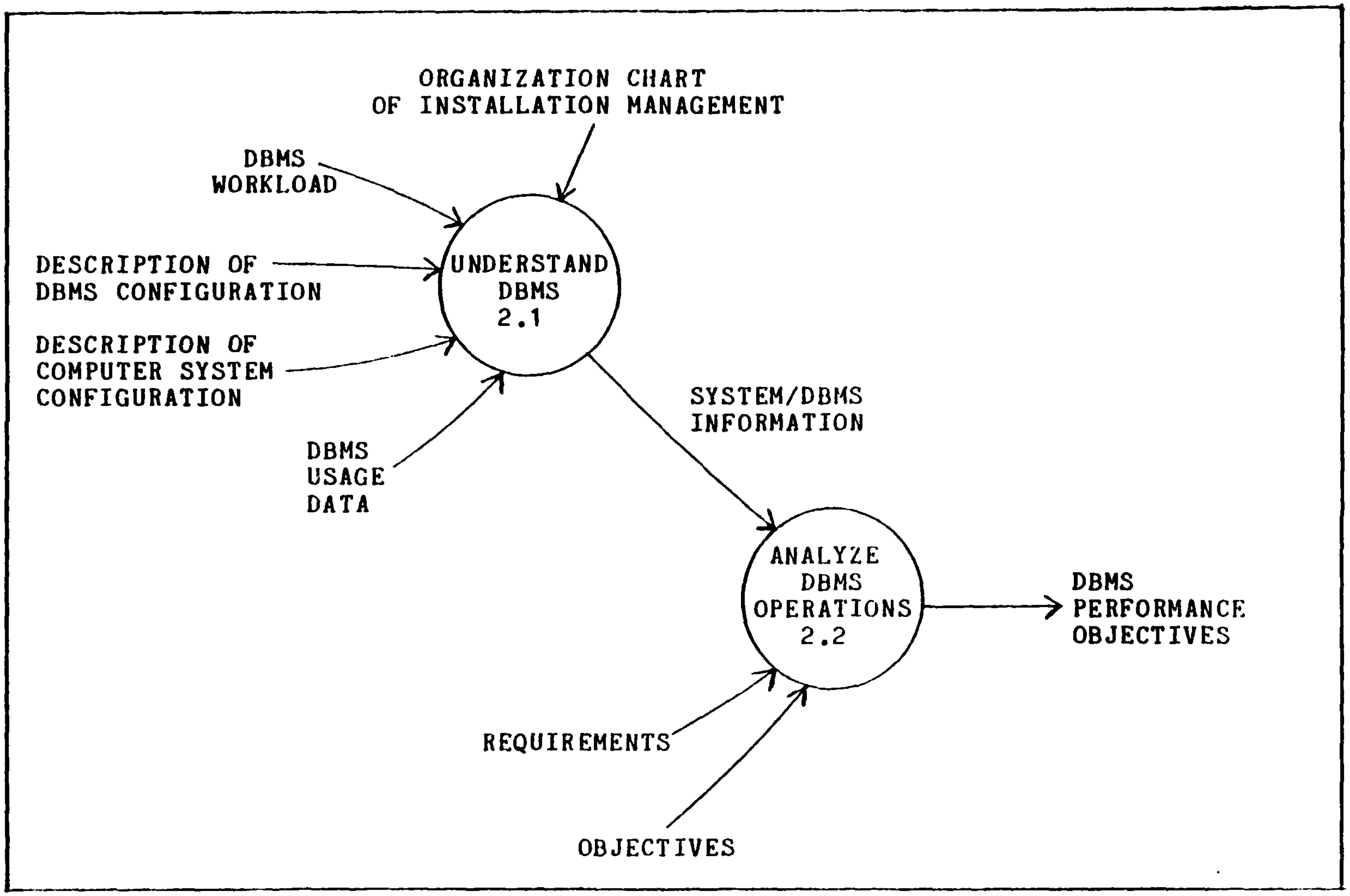

ORGANIZATION CHART OF INSTALLATION MANAGEMENT
DBMS WORKLOAD
DESCRIPTION OF DBMS CONFIGURATION
DESCRIPTION OF COMPUTER SYSTEM CONFIGURATION
DBMS USAGE DATA
UNDERSTAND DBMS 2.1
SYSTEM/DBMS INFORMATION
ANALYZE DBMS OPERATIONS 2.2
DBMS PERFORMANCE OBJECTIVES
REQUIREMENTS
OBJECTIVES

Figure II-18. Determine DBMS Objectives

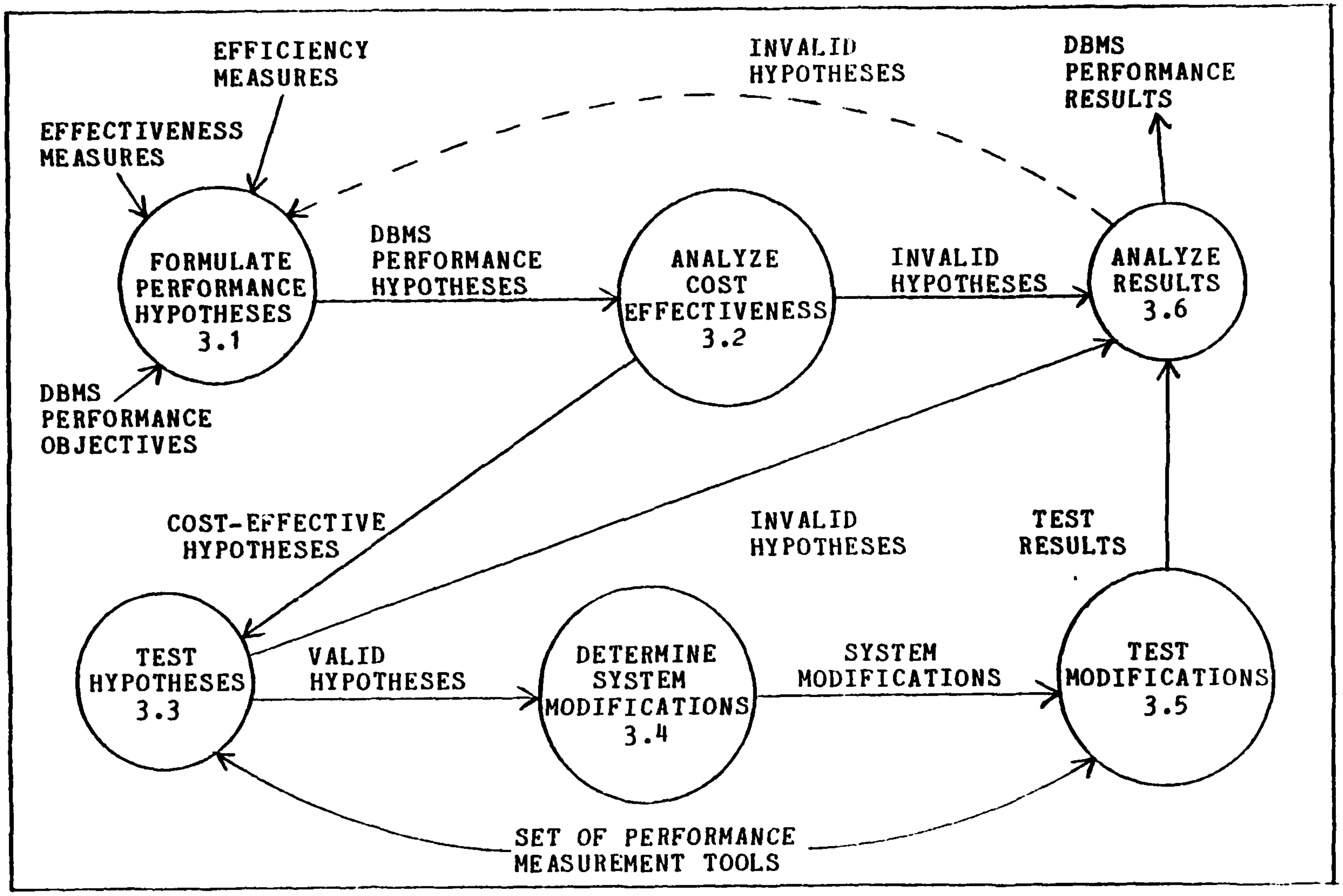


Figure II-19. Analyze DBMS Performance

collect, and how to physically and mathematically analyze the data (ref. 2:315; 5:5-9; 24:9-11; and 46:9). When these questions and their corresponding sub-questions are answered, the requirements for the performance evaluation are known, and the functional requirements of the performance measurement tools used in the performance evaluation are defined. The analysis presented in this chapter has asked these questions, and in the process of answering them, four major functional requirements of any performance tool surfaced. The four major functional requirements are:

1. It must provide a user interface.
2. It must measure the desired set of performance parameters.
3. It must be able to mathematically analyze the measurement data.
4. It must be able to present the measurement data to the user.

Based on these four requirements and the material developed in this chapter, a summary of the functional requirements for a DBMS performance monitor were developed, and they are presented in Table II-7.

TABLE II-7

<u>Functional Requirements for a DBMS Performance Monitor</u>

1.0 Establish a user interface to the monitor.

- 1.1 Allow the user to select a set of performance parameters to be measured.
- 1.2 Allow the user to specify and initiate the measurement collection process.
- 1.3 Allow the user to inspect the status of the measurement collection process.
- 1.4 Allow the user to gracefully terminate the measurement collection process before the specified collection interval has expired.
- 1.5 Allow the user to specify the types of statistical analysis to be performed on the measured data.
- 1.6 Allow the user to specify how the measurement data is to be presented.

2.0 Measure the selected set of performance parameters.

- 2.1 Create or initialize the data file and record the measurement data.
- 2.2 Map the selected set of performance parameters to a measurement source.
- 2.3 Minimize monitor artifact by measuring and recording only the selected set of performance parameters.

TABLE II-7 (Continued)

<u>Functional Requirements for a DBMS Performance Monitor</u>

- 2.4 Start the measurement collection process at the specified time.
- 2.5 Stop the measurement collection process at the specified time.

3.0 Analyze the measured values for the performance parameters.

- 3.1 Perform the necessary mathematical operations to obtain values for the performance parameters derived through calculations.
  - 3.1.1 Create an output file of the analyzed performance measurement data.
- 3.2 Perform the necessary statistical analysis.
  - 3.2.1 Create an output file of the statistical analysis data.

4.0 Present the measurement data to the user.

- 4.1 Create the performance report.
- 4.2 Print the performance report - in as many copies as the user requires.
- 4.3 Display the performance report on a terminal.

Summary

This chapter presented a detailed analysis of Computer Performance Evaluation (CPE) and its application to specific resources such as a Data Base Management System (DBMS). Part One of this chapter presented the necessary background information on DBMSs and CPE, and this provided for a common understanding of the terms and concepts used throughout this study. Part Two of this chapter developed and specified the CPE process using Data Flow Diagrams. Based on this analysis, the task of evaluating the performance of a DBMS was classified as a subset of CPE in general, and the unique DBMS performance parameters were developed. The different types of performance evaluation tools where examined for their usefulness in a DBMS performance monitor, and it was concluded that some of the existing types of performance tools could be used to gather performance information relevant to a DBMS. However, a specialized tool such as a DBMS instrumentation program/utility was also required to obtain detailed DBMS performance information. Lastly, the requirements for a generalized DBMS performance monitor were extracted and summarized in Table II-7.

The next three chapters use the functional requirements in Table II-7 as the basis for designing, implementing, and testing a DBMS performance monitor.

III. System Design

Introduction

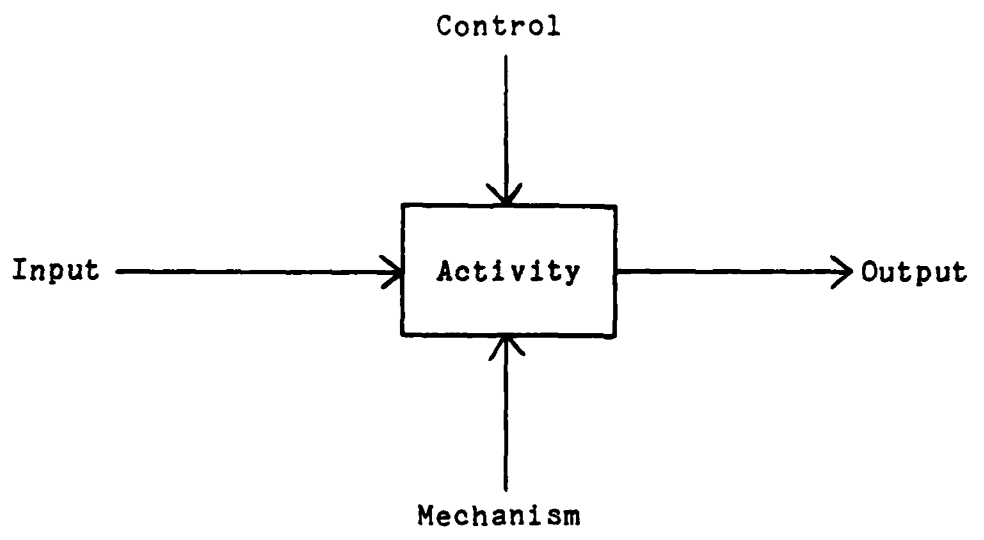
This chapter presents a generalized design for a DBMS performance monitor. The top-down design technique of stepwise refinement (ref. 30:18; 41:131; and 53:51,250) was used to develop the design, and the first step was to translate the monitor's high level (major) functional requirements into corresponding high level activities. Next, the process of stepwise refinement was continued until all the functional requirements were satisfied and all the activities had been decomposed into sub-activities that are easily implemented. Descriptions of the four high level activities of the DBMS performance monitor are presented in this chapter; however, the details of decomposing the sub-activities and their descriptions is reserved for the design documention contained in Appendix D.

In addition to describing the four high level activities, this chapter describes: the documentation technique, the operation of the performance monitor, the test plan for the design, and the use of the design to implement a DBMS performance monitor.

Documentation Technique

The Structured Analysis and Design Technique (SADT) developed by SofTech Corporation was used to document the generalized design. Other techniques, such as structured english, could have been used; however, the SADT technique was chosen because it contains specific methods for showing the activities that must be accomplished by a design. By showing the system activities, the SADT technique specifies what has to be accomplished before the details of how it is accomplished are introduced. Therefore, the implementation details are forced to the lower levels of the problem solution, and the design documentation does not resemble programming logic. This is an important capability for a documentation technique since the concept of forcing the implementation details to the lower levels of actual program development is a major goal in developing software systems (ref. 41:131 and 53:5).

The mechanics of an SADT activity diagram are shown in Figure III-1. The box represents the activity to be performed, and the arrows represent the data associated with the activity. An additional advantage of SADT activity diagrams is their ability to specify control and mechanism inputs. In a DBMS performance monitor, controlling the measurement activity and the mechanism used to measure performance parameter values are important concepts, and for this reason, how they affect the system design must be reflected in the design documentation.



Input - Data consumed or transformed by the activity
Control - Data which controls or constrains the activity
Output - Data produced by the Activity
Mechanism - Processor or tool used to help accomplish the activity

Figure III-1. SADT Activity Diagram

The SADT documentation technique also includes a data diagram. In an SADT data diagram, the box represents the data, and the arrows represent the activities associated with the data. Data diagrams were not used in the design of the DBMS performance monitor.

Documentation for SADT diagrams is presented in the form of a reader kit consisting of a diagram index, the activity and/or data diagrams, and a data dictionary. A complete SADT reader kit for a generalized DBMS performance monitor is contained in Appendix D.

Design Description

The design presented in this chapter is based on the use of software tools to accomplish as many of the functional requirements of a DBMS performance monitor as possible - especially in the area of measuring and recording values for the performance parameters. This strategy does not restrict the generality of the design because no specific software tools have been included in it. Each of the activities represented in the SADT diagrams could be accomplished by software tools, specially designed programs, or a combination of software tools and specially designed programs.

As previously stated, the top level SADT diagram was obtained by directly translating each of the four major functional requirements of a DBMS performance monitor into corresponding activities. The top level diagram derived by the translation is shown in Figure III-2, and this figure is a reproduction of page D-6 from the SADT reader kit of Appendix D. An important aspect of this figure is the method of communication between the interconnected activities. As shown in figure III-2, the user interface activity serves to collect a user's input to the DBMS performance monitor, and it produces a set of commands that are subsequently used to control the measurement, analysis, and presentation activities. This method of communication between the activities is important for

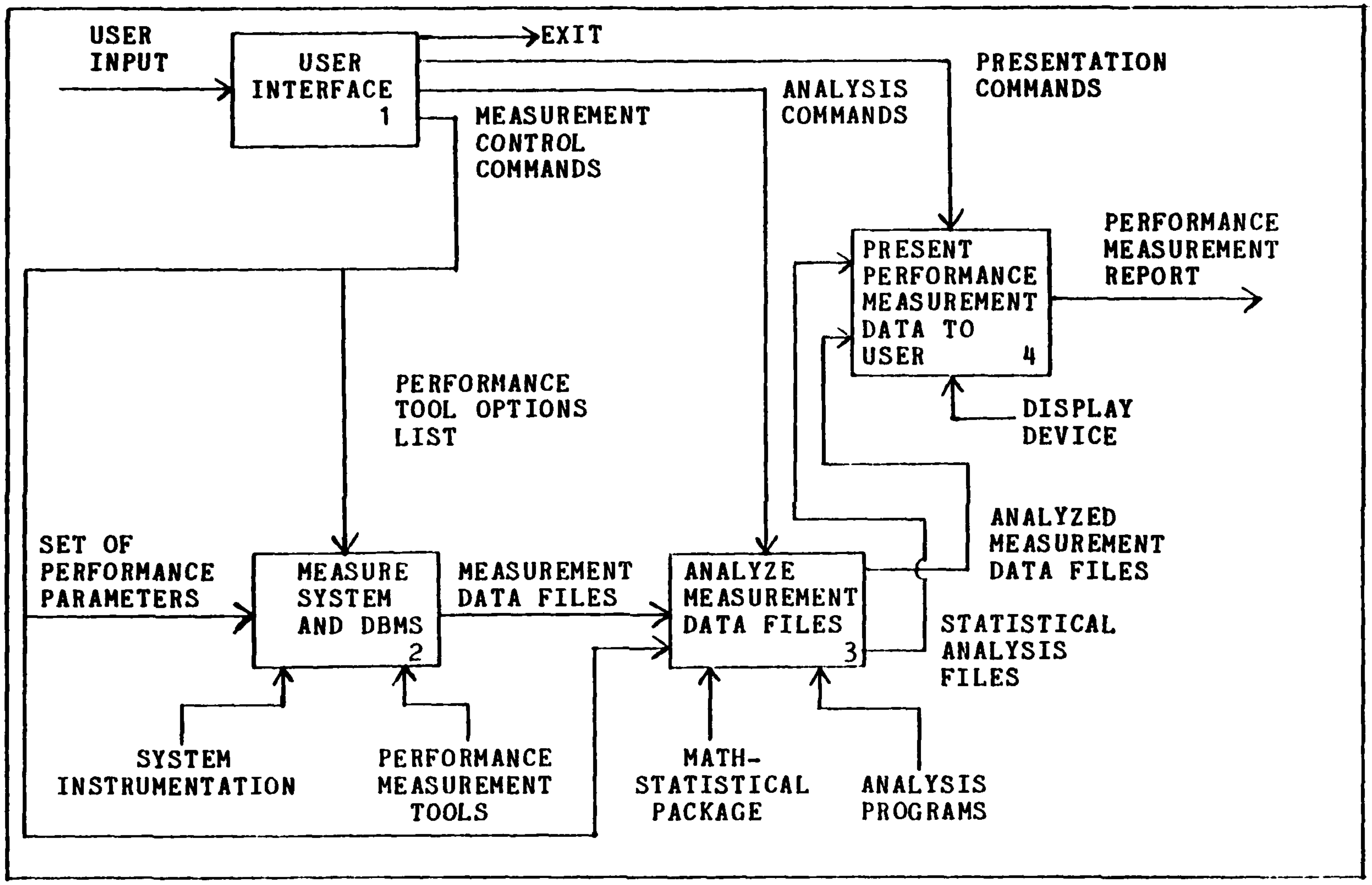

USER INPUT
USER INTERFACE 1
EXIT
MEASUREMENT CONTROL COMMANDS
ANALYSIS COMMANDS
PRESENTATION COMMANDS
PERFORMANCE TOOL OPTIONS LIST
PRESENT PERFORMANCE MEASUREMENT DATA TO USER 4
PERFORMANCE MEASUREMENT REPORT
DISPLAY DEVICE
SET OF PERFORMANCE PARAMETERS
MEASURE SYSTEM AND DBMS 2
MEASUREMENT DATA FILES
ANALYZE MEASUREMENT DATA FILES 3
ANALYZED MEASUREMENT DATA FILES
STATISTICAL ANALYSIS FILES
SYSTEM INSTRUMENTATION
PERFORMANCE MEASUREMENT TOOLS
MATH-STATISTICAL PACKAGE
ANALYSIS PROGRAMS

Figure III-2. Top Level Design Diagram

three reasons. First, it allows the user to completely specify an entire measurement session or just certain activities of the DBMS performance monitor. For example, a user may want to specify a measurement session that executes late at night and be able to receive the results the first thing the next morning. Alternately, a user many want to specify just the measurement activity and delay the specification and execution of the analysis and presentation routines until some later time. Second, this method is important because it does not dictate how the activities are actually implemented on a target system. This allows the flexibility of using software tools and/or specially designed programs. Lastly, this method is important because it supports the software development technique of iterative enhancement.

Iterative enhancement is the technique of selecting a subset of the problem and designing and implementing this subset first (ref. 4:121-127 and 53:54-55). In this way, a running system (although limited in function) is produced earlier, and the early system can be easily evaluated and changed. After the early system is running satisfactorily, the process is repeated using successively larger subsets until the entire system has been developed and tested.

The technique of iterative enhancement can be easily applied to the DBMS performance monitor. For example, the measure system and DBMS activity of Figure III-2 could be implemented first with the goal of producing measurement data as quickly as possible.

A descriptive summary of each of the four activity diagrams shown in Figure III-2 is provided below.

User Interface. This activity satisfies functional requirements numbers 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, and 1.6 from Table II-7. It accepts the user's input to the DBMS performance monitor, and it uses the input to construct three command sets for controlling a measurement session. The commands sets serve as the interface to the three other activities shown in Figure III-2. The contents of each command set are briefly described below.

Measurement Control Commands. This command set controls the measuring and recording of values for the performance parameters, and the individual commands within the command set control three aspects of a performance measurement session. First, the commands select the performance measurement tools required to measure the desired set of performance parameter values. Second, the commands specify operating conditions for the performance tools such as start time, stop time, data file names, etc.. Lastly, the commands initiate the performance tools to begin monitoring the system. In the event a selected performance tool can not be automatically initiated because it requires human intervention(i.e. a hardware monitor), the commands initiate the printing or display of a set of instructions for connecting the performance tool to the measured system.

Analysis Commands. This command set controls the mathematical analysis of the data recorded by the performance tools. The individual commands within this command set select the necessary analysis programs or math-statistical packages and schedule them for execution.

Presentation Commands. This command set controls the presentation of the performance measurement data to the user of the monitor. The individual commands within this command set select the required formatting programs and schedule them for execution. Additionally, the display device for the data is selected.

Measure System and DBMS. This activity satisfies functional requirements numbers 2.0, 2.1, 2.2, 2.3, 2.4, and 2.5 from Table II-7. It utilizes the performance measurement tools to measure values for the set of performance parameters specified by the user of the monitor. The measurement activity is controlled by the measurement control commands generated by a user's input to the user interface activity. The values measured by this activity are recorded in measurement data files for later analysis and presentation to the monitor user.

<u>Analyze Measurement Data Files</u>. This activity satifies functional requirements numbers 3.0, 3.1, 3.1.1, and 3.2 from Table II-7. It uses the measurement data files produced by the performance tools to calculate values for the performance parameters that are not directly measureable and to produce statistical analysis of the data. The analysis activity is controlled by the analysis commands generated by a user's input to the user interface activity. The analyzed measurement data and any required statistical analysis data are stored in files for later presentation to the monitor user.

<u>Present Performance Measurement Data</u>. This activity satisfies functional requirements numbers 4.0, 4.1, 4.2, and 4.3 from Table II-7. It presents the analyzed measurement data and any statistical analysis data to the user of the DBMS performance monitor in the form of a performance measurement report. The presentation activity is controlled by the presentation commands generated by a user's input to the user interface activity.

Monitor Operation

The design presented in this chapter implies each of the four major activites of Figure II-2 occur in a sequential manner. For an individual measurement session, the sequential order of processing must be adhered to; however, the sequential nature of an individual measurement session does not preclude the operation of several measurement sessions in parallel. The limiting factor to parallel operation is the increased system interference caused by the parallel execution of the performance measurement tools. Generally speaking, there should be no need to have parallel measurement sessions, but this should not limit the flexibility of the design.

Test Plan

A formal test plan is used to test the validity of a design as well as any implementations based on the design. For a test plan to be useful, it must be developed with the intent of finding errors (ref. 37:5). Two methods commonly used to test software are known as "black-box" and "white-box" testing (ref. 37:8-11 and 41:292-293). Other techniques for developing test cases and testing programs do exist; however, the majority of these techniques fall into the generic classes of "black-box" and "white-box" testing. Examples of these techniques are

Logic Coverage, Equivalence Partitioning, and Boundary Value Analysis (ref. 41:305-311).

"Black-box" test procedures derive their test cases solely from the requirements and specifications of the system without taking advantage of any knowledge about the internal structure of the software. On the other hand, "white-box" test procedures derive their test cases from an examination of the program logic.

At this point, "white-box" testing cannot be used; only the design of the DBMS performance monitor has been specified. However, the "black-box" testing method and the functional requirements of Table II-7 can be used to derive a set of test cases for the generalized design presented in this chapter and Appendix D. An example test plan module is shown in Figure III-3, and a complete test plan for the proposed design is contained in Appendix D.

Using the Design

The complete design presented in Appendix D shows the activities and sub-activities that must be accomplished in order to develop a DBMS performance monitor meeting the functional requirements of Table II-7. The next step in the problem was to use the design to implement a DBMS Performance Monitor for a specific host computer system and DBMS. The activities shown in the SADT diagrams do not necessarily reflect a one-to-one mapping

REQUIREMENT: 1.1 - Allow the user to select a set of performance parameters.

TEST CASE(S):

1. Select no parameters.
2. Select all parameters.
3. Select the pre-defined Software Engineer's subset of parameters.
4. Select a specialized subset of parameters.
5. Restart the selection process.

EXPECTED RESPONSE:

1. The default set of parameters is selected.
2. The complete set of measurable parameters is selected.
3. The Software Engineer's subset is selected.
4. Only the special subset is selected.
5. The currently selected set of parameters is deleted and a new set is started.

RESULTS:

CASE 1. -	PASS:____	FAIL:____	DATE:____________
CASE 2. -	PASS:____	FAIL:____	DATE:____________
CASE 3. -	PASS:____	FAIL:____	DATE:____________
CASE 4. -	PASS:____	FAIL:____	DATE:____________
CASE 5. -	PASS:____	FAIL:____	DATE:____________

TESTED BY:______________________________

REMARKS:

Figure III-3. Test Plan Example

from the activity diagram to a computer program or a software tool. Therefore, a study of the software tools and general capabilities of the target system needs to be performed, and the results of the system study are presented in the next chapter.

Summary

This chapter introduced the SADT design documentation technique, and this technique was used to present a generalized design for a DBMS performance monitor. The four major activities of a DBMS performance monitor were defined as,

1. User Interface
2. Measure System and DBMS
3. Analyze Measurement Data Files
4. Present Performance Measurement Data

and each of these activities were examined in detail to specify the interface between them. The methodology for developing a test plan for the design was presented, and an example test plan module was shown. Lastly, an approach for using the design was presented, and the results of this approach are presented in the next chapter.

IV. VAX 11/780 Implementation

Introduction

This chapter discusses how the generalized design for a DBMS performance monitor was implemented on a Digital Equipment Corporation (DEC) VAX 11/780 computer for the TOTAL DBMS. As with most software systems, there is more than one way to realize an implementation, and a DBMS performance monitor is no exception to this rule. Therefore, the implementation plan presented in this chapter is just one of many possible implementations.

The topics discussed in this chapter include: the approach used to implement the monitor, the operating characteristics of the VAX 11/780 computer and TOTAL DBMS, the performance tools available on the VAX 11/780, the implementation options, and the implementation plan. The discussions on the operating characteristics of the VAX 11/780 and TOTAL DBMS are included in this chapter to help illustrate the importance of understanding the system to be evaluated, and they are useful for relating the results of the DBMS performance monitor to specific aspects of the VAX computer and TOTAL DBMS.

Implementation Approach

The first thing performed during the implementation stage was a study of the VAX 11/780 computer system and the TOTAL DBMS. The five goals of the system study were to:

1. Understand the hardware and software configurations of the VAX 11/780.

2. Understand the TOTAL DBMS and how it works on a VAX 11/780 computer.

3. Determine the performance tools available for the VAX computer and the TOTAL DBMS.

4. Determine the capability of each performance tool to measure the system, analyze the measured data, and present the data to a user of the tool.

5. Determine what other types of software tools were available for performing tasks such as data analysis, statistical analysis, and developing "user friendly" interfaces.

After the system study was completed, the next step was to apply the information to the generalized design, and this involved selecting the useful tools, determining any additional computer programs to be developed, and specifying the communication interfaces between the four major activities of the DBMS performance monitor. The third step in the implementation was to develop the additional programs. The program development included analyzing the data structures required by the programs, designing the programs, coding the programs, documenting the programs, and individually testing each of the programs. The final implementation step was to perform a system test of the DBMS performance monitor in accordance with the test plan developed for the generalized design of the monitor.

The system study and detailed design steps are presented in this chapter, and the program development and testing steps are presented in Chapter 5. The system testing step is presented in Chapter 6.

VAX 11/780 Configuration

Figure IV-1 contains a hardware configuration diagram of the VAX 11/780 computer system used in this study (ref. 15:7-18). The system contains 2.5 megabytes of main memory, two RK07 disk drives (each having a storage capacity of 28 megabytes), one tape drive, six terminals, and one line printer. All peripheral devices are connected to the processor through the UNIBUS, and the UNIBUS allows data to be transferred in one of two ways. The data can be transferred in a block as a DMA transfer, or it can be transferred on a byte-by-byte basis through program interrupts (ref. 15:16).

The operating system used on the computer was version 3.4 of the VAX/VMS operating system. The VMS operating system employs a virtual memory management

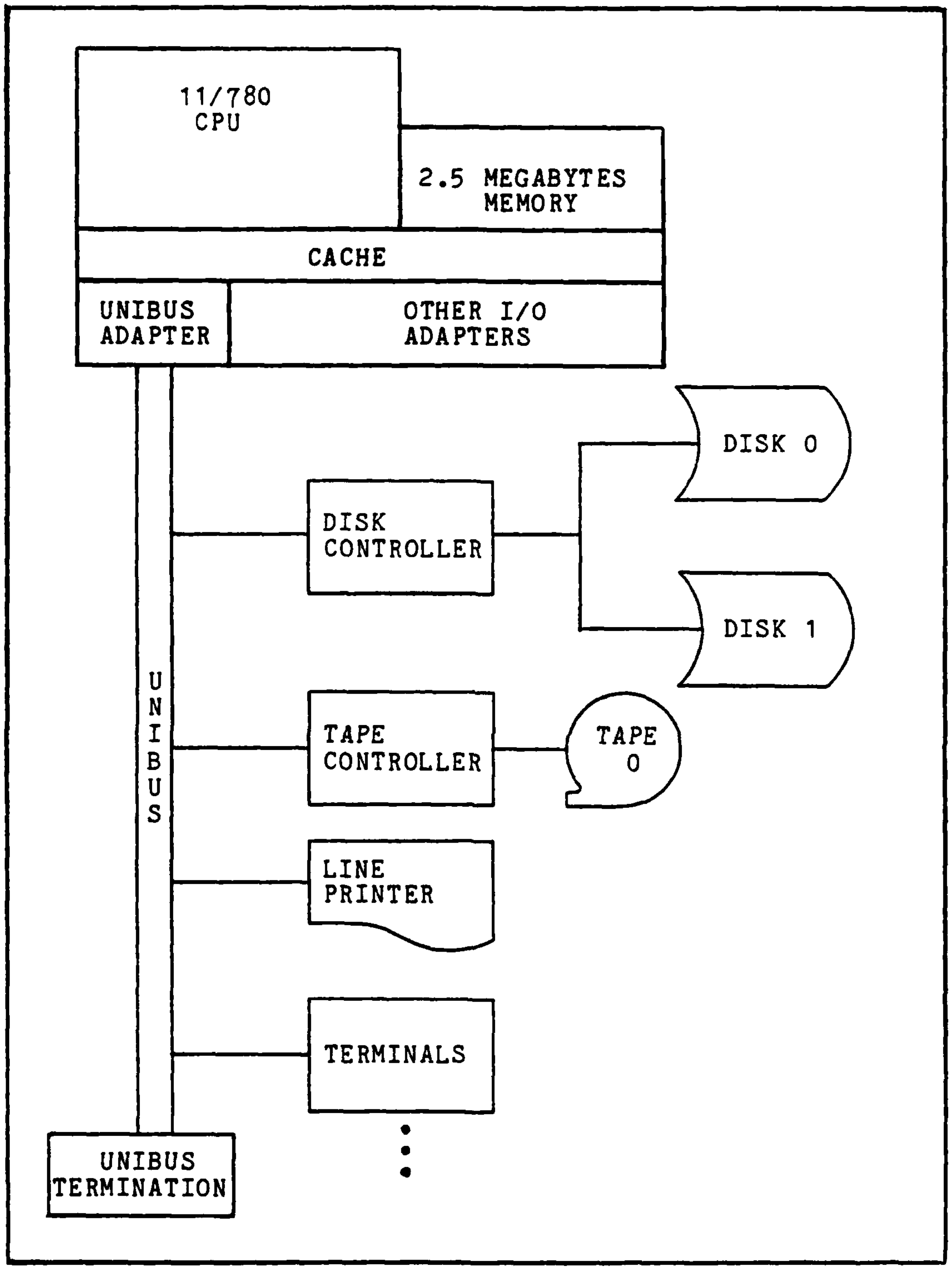


Figure IV-1. VAX 11/780 Hardware Configuration

scheme of demand paging using a fixed page size of 512 works (1K bytes), and the principles of its operation are briefly described below (ref. 17:Chapter 2,3 and 25:Appendix A).

Individual user jobs are referred to as a process within the operating system, and each active process has a working set of primary memory pages associated with it. The working set refers to the number of pages a process has resident in main memory. When enough pages of memory have been allocated to fulfill a processes' minimum working set requirements, it joins all the other active processes in the system's balance set. The balance set is all the processes waiting to be scheduled for the CPU or I/O operations. Since a processes' working set may be smaller than the overall memory space requirements, memory references to regions of the process not currently contained in the working set will occur periodically, and this causes a hardware page fault. A page fault event may force the VMS operating system to select a page currently in the working set for removal in order to make room for the new page being referenced.

A page selected for removal from the working set is not actually removed from main memory. It is placed on either the free page list or the modified page list. The free page list keeps track of those available pages which were not altered during their use, and the modified page list keeps track of those pages that were altered during their use. These two lists make up a shared page cache, and before the VMS operating system initiates a disk read to satisfy a page fault, these lists are checked to determine if the required page is already in memory. If the required page is found, it is removed from the list and connected back into the processes' working set saving the time needed to perform a disk read.

If the required page is not found, space is taken from the free page list to store the new page read from the disk. If the free page list is empty, space must be taken from the modified page list. However, pages on this list have been altered and must be saved on the disk before the new memory page is read from the disk into the memory space currently held by the the modified page. This situation could cause considerable overhead, and the VMS operating system tries to prevent this situation from occuring by writing the modified pages to disk whenever the system is idle or the list exceeds a certain length. Modified pages saved in this manner are moved to the free page list for later use.

TOTAL DBMS

The Data Base Management System (DBMS) used in this study is Version 2.1 of the TOTAL DBMS marketed by CINCOM

Systems Inc.. The TOTAL DBMS uses the network data model for storing and retrieving information (ref. 8:1-3), and its basic principles of operation on a VAX 11/780 computer are described below:

Data Base Structure (ref. 8:1-3). A TOTAL data base consists of a group of data sets (files). There are two types of data sets. The first type is called a master data set. Master data sets are independent, and the logical records within a master data set can be directly accessed using the control key. The second type of data set is called a variable data set. Variable data sets are dependent, and they must be attached to a master data set. Logical records within a variable data set are chained to a unique master record in a related master data set. The chaining provides the access paths to the data, and these paths are referred to as linkpaths. A master data set can have more than one variable data set attached to it, and a variable data set can be attached to more than one master data set. Therefore, multiple access paths can exist from a single master data set to multiple variable data sets, and from multiple master data sets to a single variable data set.

The structure of a TOTAL data base relates to the Data Base Task Group (DBTG) definition of a network model data base in the following way. Set types defined for a data base would use records from a master data set as owner records and records from a variable data set as member records. There is no direct correspondence to a connection record type because a TOTAL data base defines fields within master and variable data set records to serve as the connection mechanism. These fields are referred to as the linkpath.

Figure IV-2 illustrates a simple TOTAL data base with one master data set and one variable data set. The control key is used to select a unique master record from the master data set. After the master record has been selected, the linkpath field within the master record is used as the access path to a group of chained records in the variable data set. For example, the master record could contain all the personal and employee data for a car salesman, and the chained group of variable records could represent the information on all the cars the salesman has sold. The control key for the master data set could be any unique identifying information about the salesman such as employee number, social security number, etc..

<u>DML Commands</u>. The Data Manipulation Language (DML) commands available in the TOTAL DBMS are (ref. 8:2-26):

<u>Serial Processing Functions</u>. These functions are used to process records one-by-one according to their

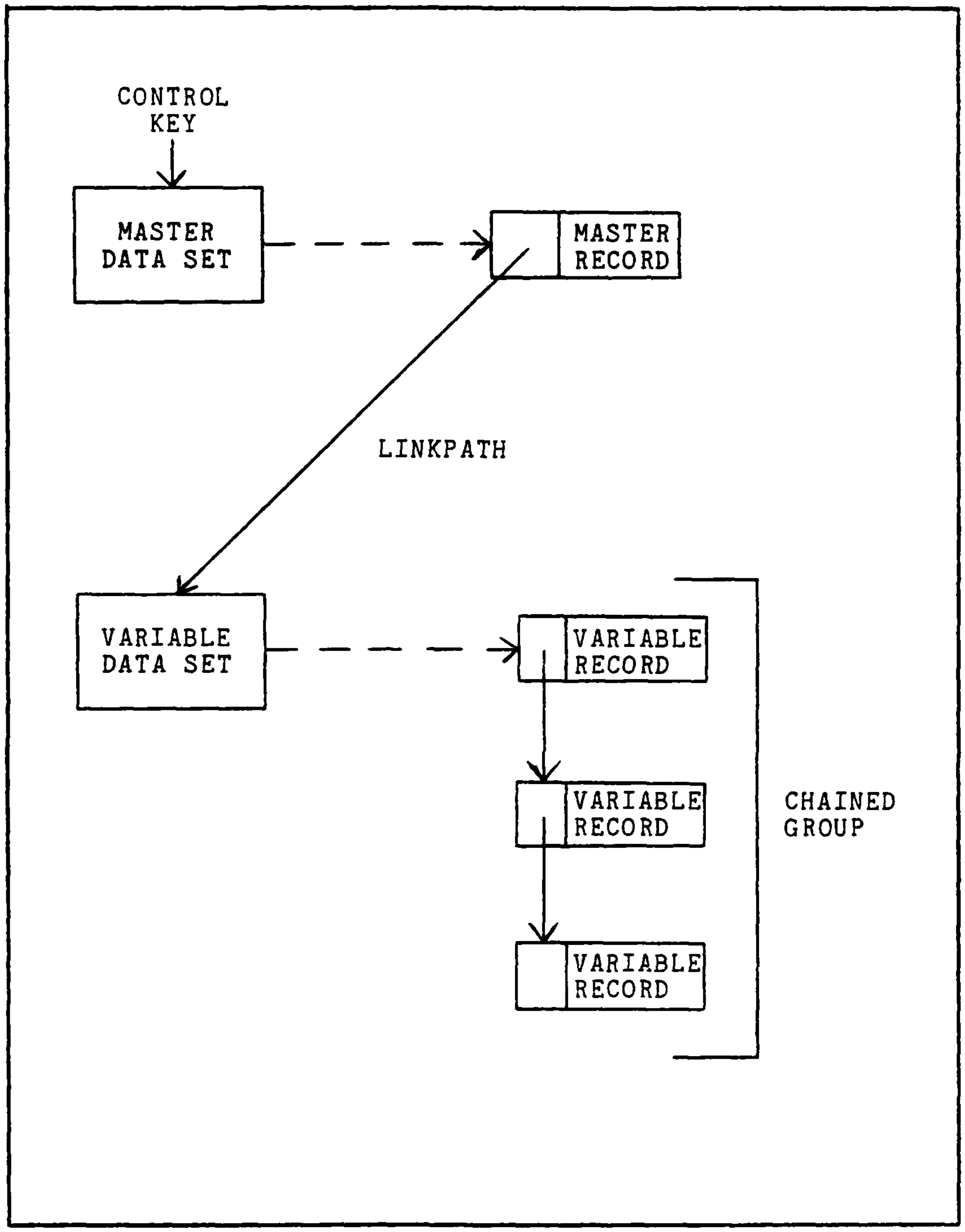


Figure IV-2. TOTAL Data Base Structure

physical sequence in a data set. Serial processing is accomplished by repeating the same function.

1. RDNXT - Serially read a master or a variable entry data set.

Master Data Set Functions. These functions are used to process a record from a master data set.

1. READM - Read a master record.
2. WRITM - Write a master record.
3. ADD-M - Add a master record.
4. DEL-M - Delete a master record.

Variable Data Set Functions. These functions are used to process a record from a variable data set.

1. READV - Read a variable record along the forward direction of a variable record chain.
2. READR - Read a variable record along the reverse direction of a variable record chain.
3. READD - Read a variable record directly by specifying its position.
4. WRITV - Write the variable record retrieved by the preceding read.
5. ADDVC - Add a variable record to the end of the chain.
6. ADDVB - Add a variable record before the one retrieved by the preceding read.
7. ADDVA - Add a variable record after the one retrieved by the preceding read.
8. DELVD - Delete the variable record retrieved by the preceding read.

Special Functions. These functions are used to control the processing of specialized DBMS operations.

1.	SINON	- Sign-on to the DBMS.
2.	SINOF	- Sign-off the DBMS.
3.	RQLOC	- Request the home location of a master record.
4.	WRITD	- Write a master or variable record directly into a specific location.
5.	QUIET	- Check point the data base and log device.
6.	LOADD	- Load a data base descriptor.
7.	FREEF	- Free held records for a file.
8.	FREEX	- Free all held records for this program.

In chapter two, four general types of DML statements were defined. These four types were defined as retrieval, storage, control, and special purpose. The four types were grouped based on their general DBMS function so the performance evaluation aspects of these DBMS functions could be more closely evaluated. In keeping with this idea, the TOTAL DML commands have been grouped into the four general types, and the results of this grouping are shown in Table IV-1.

DBMS Configuration. Figure IV-3 shows how the TOTAL DBMS works on a VAX 11/780 computer system (ref. 8:2-2). On a VAX host, the TOTAL DBMS is executed as a background batch job, and the DBMS applications programs are executed either as batch or interactive jobs. The

TABLE IV-1

Generalized Grouping of the TOTAL DML Commands

Retrieval	Storage	Control	Special Purpose
RDNXT	WRITM	SINON	RQLOC
READM	ADD-M	SINOF	WRITD
READV	DEL-M	QUIET	LOADD
READR	WRITV		FREEF
READD	ADDVC		FREEX
	ADDVB		
	ADDVA		
	DELVD		

applications programs communicate with the TOTAL DBMS using subroutine calls to the subroutine DATBAS, and the parameters of the subroutine call make-up the TOTAL DML command. The DATBAS subroutine communicates with the TOTAL DBMS through the VAX Mail Facility to send/receive messages, data, and commands to/from the TOTAL DBMS. When there is no mail to TOTAL, the TOTAL DBMS hibernates and waits for incoming mail.

An example of the complete sequence of steps required to retrieve a logical record from a TOTAL data base is outlined below. Figure IV-3 contains the step numbers to allow the sequence to be easily traced on the configuration diagram.

1. The user applications program sets up the parameters for the DML command and calls the DATBAS subroutine.

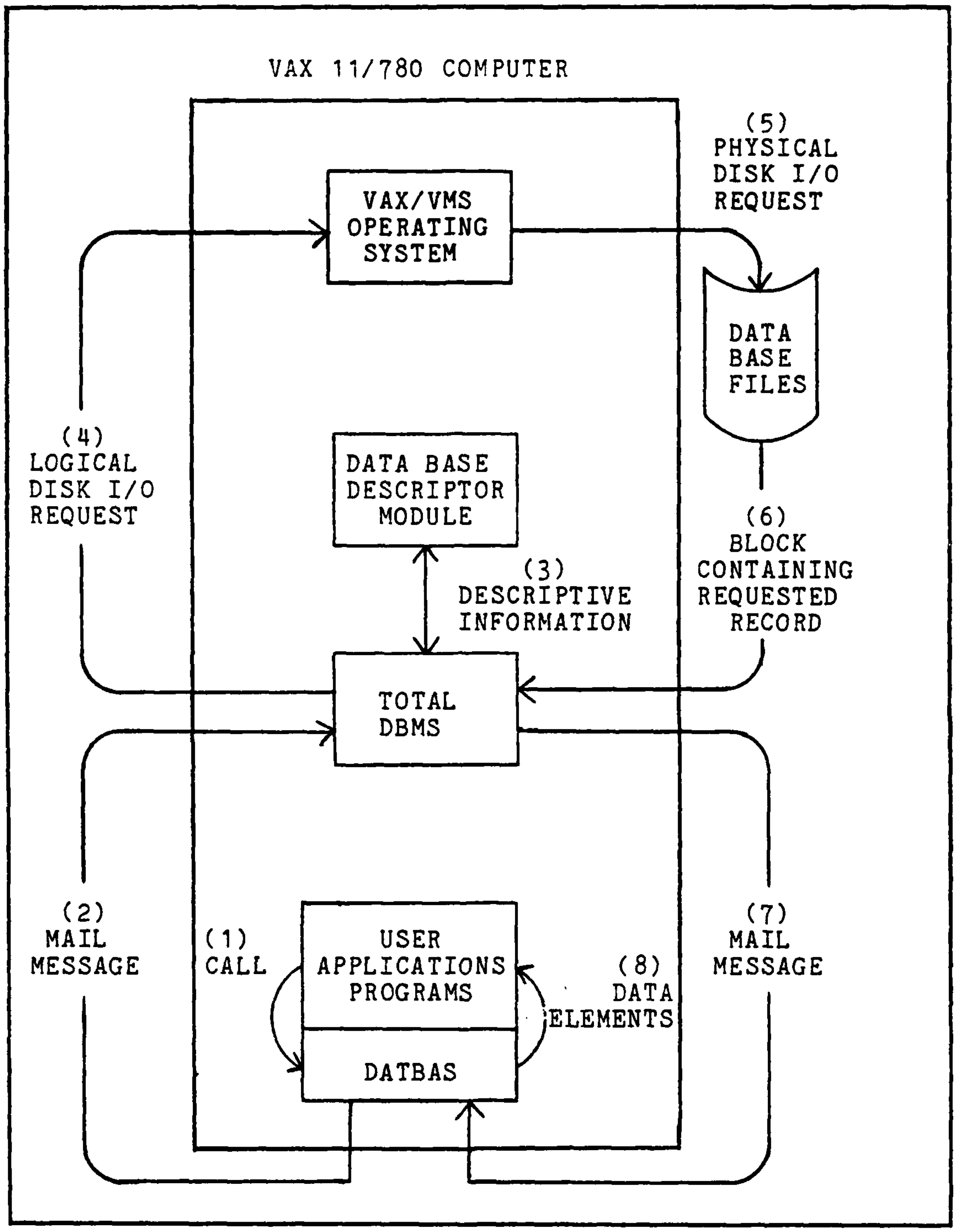

VAX 11/780 COMPUTER
VAX/VMS OPERATING SYSTEM
(5) PHYSICAL DISK I/O REQUEST
DATA BASE FILES
(4) LOGICAL DISK I/O REQUEST
DATA BASE DESCRIPTOR MODULE
(6) BLOCK CONTAINING REQUESTED RECORD
(3) DESCRIPTIVE INFORMATION
TOTAL DBMS
(2) MAIL MESSAGE
(1) CALL
USER APPLICATIONS PROGRAMS
DATBAS
(8) DATA ELEMENTS
(7) MAIL MESSAGE

Figure IV-3. VAX-TOTAL DBMS Configuration

2. The DATBAS subroutine formats a mail message and sends it to the TOTAL DBMS.

3. The TOTAL DBMS receives the mail message and accesses the data base descriptor module (schema) to obtain descriptive information about the logical record and its data elements.

4. The TOTAL DBMS requests a logical disk I/O be performed by the VMS operating system.

5. The VMS operating system translates the logical I/O request into a physical disk I/O request.

6. The VAX I/O hardware performs the physical I/O from the TOTAL data base files (data sets) and places the data in a data buffer defined by the TOTAL DBMS.

7. The TOTAL DBMS extracts the required data elements from the disk block and sends them via a mail message to the DATBAS subroutine.

8. The DATBAS subroutine places the data element values in the data area of the user program and returns control the the user program.

Relationship to DBMS Architectural Model. The DBMS architectural model presented in Chapter 2 does apply to the TOTAL DBMS. The TOTAL DBMS allows each user to define an external level view by using a schema definition in the applications program. The data base descriptor modules

(DBMOD) make up the conceptual level view of all the data bases, and the DBMODs are used to provide the schema capability to end-users at the external level. Additionally, the DBMODs are used at the internal level to present a stored record interface to the VAX/VMS access method.

Potential Areas for Performance Problems. The complexity of Figure IV-3 graphically illustrates how important it is to understand how the DBMS operates in the VAX/VMS environment before undertaking a performance evaluation effort. This figure shows at least five areas where performance problems could potentially occur. These five areas are: the overhead involved in making calls to the DATBAS subroutine, the VAX Mail facility, the VAX/VMS access method (translating the logical disk request to a physical disk request), the physical I/O system, and the TOTAL DBMS buffer pools.

System Performance Tools.

Five performance tools were found to exist on the VAX 11/780 computer. In addition to these tools, the TOTAL DBMS provides a DBMS log facility. An additional performance tool, VAX-11 SPM (Software Performance Monitor) is available from DEC, but it was not available on the target system. The general capabilities of each tool are described below, and Appendix E contains a detailed look at each of the tools. The tables contained in Appendix E list the set of performance parameters each tool is capable of measuring, a brief description of each parameter, and the parameters used for a DBMS performance monitor.

<u>VAX Monitor Utility</u> (ref. 21:Chapter 12). The monitor utility is a software monitor used to obtain information on the performance of the VMS operating system. The monitor utility collects data on a time-driven basis, and the sampling interval between data collection events can be set between the range of 1 to 9,999,999 seconds. It has the capability of monitoring nine classes of system wide performance data and producing a variety of summary reports. The nine classes of performance data are: DECNET, FCP, IO, LOCK, MODES, PAGE, POOL, PROCESSES, and STATES. Of these nine, four are useful for the development of a DBMS performance monitor, and the four classes are briefly described below.

1. IO - Monitors the I/O system and produces values for the performance parameters such as: rate of disk and tape I/O operations, page fault rate, page read rate, size of the free and modified page lists, etc..

2. MODES - Monitors the time spent in each of

the seven processor modes such as: processor idle, supervisor mode, user mode, etc..

3. STATES - Monitors the number of processes in each of the fourteen scheduler states such as: waiting for CPU, suspended, waiting for a free page of memory, waiting for a page fault, etc..

4. FCP - Monitors the VAX/VMS file system and produces values for performance parameters such as: disk space allocation rate, file open rate, file creation rate, rate at which CPU time is used by the file system, etc..

The monitor utility can produce three types of output, and it has the capability to analyze the data before producing the output. The monitor utility can display the output on a terminal, generate a binary data file of non-analyzed data, or generate an ASCII formatted print file of the analyzed data. The output options are not mutually exclusive providing the capability to produce several types of output simultaneously.

VAX-11 SPM (ref. 16:). The VAX-11 SPM (Software Performance Monitor) utility is a software monitor used to collect and report performance statistics for VAX/VMS computers. It is similar to the monitor utility, but it provides an extended set of capabilities. For example, it can: measure CPU-I/O overlap, measure where the overall system is spending its CPU time, trace specific VMS events, and produce histograms of where user programs are spending their CPU time.

<u>VAX Accounting Utility</u> (ref. 21:Chapter 1). The accounting utility has the capability to read the system job accounting file, select records from the file, and produce a summary report or a new data file. The summary report can be tailored to meet many needs because the capability exists to select the desired data items and group them by a list of summary keys. Summary reports are produced only in the form of an ASCII formatted print file. However, new data files can be produced as either binary files or ASCII formatted print files. The accounting utility can produce performance values for individual user jobs such as: elapsed time, total CPU time, total page faults, total disk and tape I/Os, total number of disk and tape volumes mounted, etc..

<u>VAX SYE Utility</u> (ref. 21:Chapter 17). The SYE utility can selectively extract records from the system error log file and generate summary reports in an ASCII print file format. The SYE utility can produce performance values for the error rates and reliablity of the CPU, memory, disk drives, and other peripheral devices.

VAX _Run_ _Time_ _Library_ (ref. 20:5-24). The run time library provides four procedures that can be used to instrument a process (applications program) and test its performance. The four procedures can be used to obtain the processes' elapsed time, CPU time, I/O counts, and the number of page faults. These procedures are easy to use, but they are constrained by two factors. First, they provide only a limited amount of information, and second, they can only be used to obtain information on the specific process being executed. Performance information on other active processes is not available.

VAX _System_ _Services_ _Library_ (ref. 18:124-132). The system services library provides the SYS$GETJPI procedure which provides access to a wide range of performance information on a user process. Additionally, it has the capability to retrieve information about more than one specific process. The system service procedure is a more powerful tool to instrument a process (applications program) since it can simultaneously measure the performance of several processes. For example, it could be used to simultaneously measure the performance of the DBMS and all user processes of the DBMS. One drawback of this procedure is that it does not provide data analysis capabilities.

The system services library also provides the

SYS$GETTIM procedure which provides access to the system clock to a resolution of 100 nanoseconds. This procedure is useful for accurately measuring the elapsed time between events.

<u>TOTAL DBMS Log Facility</u> (ref. 9:Chap 7 and 8). The TOTAL logging facility provides the capability to log before images, DBMS functions, or both. Log information can be recorded on either tape or disk media, and the capability exists to define a two file flip-flop approach. The information is recorded in the log file in variable length records inside of a fixed size block, and the block size is established at data base generation time.

Typically, the information contained in a DBMS Log file is useful for determining performance parameters such as DBMS throughput, the arrival rates of DML commands, the distribution of the arrival rates, etc. (ref. 2:319). Unfortunately, the log file provided by the TOTAL DBMS is specifically designed for recovery purposes. Therefore, the TOTAL DBMS only logs storage and control DML commands. All data pertaining to retrieval and special purpose DML commands is not recorded. Additionally, the log file does not contain the time information is recorded; it only records the date. This means the log file can't be used to calculate the arrival rate or distribution of the arrival rate for the DML storage operations (useful statistics for constructing probabilistic models of DBMS processing). For thes reasons, the TOTAL DBMS Log Facility was not use .e DBMS performance monitor.

Data Analysis Tools

Only one data analysis tool was available for the VAX 11/780 development system. This was the Haessle STAT package (ref. 40:). It was not operational on the system, and an evaluation of the user manual showed it was limited in its capabilities. It can not handle alpha-numeric data; therefore, it could not be used to select data based on alpha-numeric values such as job/process name, date, time, etc..

The capability existed to generate data tapes for use on a CYBER computer system and the SPSS software package. This tool provided the necessary data selection capabilities based on alpha-numeric values; however, it could not be used for "on-line" analysis. It required human intervention to create the tape, prepare a CYBER job to analyze the data on the tape, run the job, and pick-up the results.

Set of Measurable Performance Parameters

Of the 104 performance parameters developed during the system requirements stage (ref. Table C-1), 66 were capable of being measured on the VAX 11/780 by using a combination of the Monitor Utility, Accounting Utility, SYE Utility, and System Services Library. Of the 38 remaining parameters, 10 were not measured due to the lack of a hardware monitor, 11 were re-defined by VAX unique parameters, and 17 were not measurable. Additionally, eight parameters unique to the VAX 11/780 memory and I/O system management were added to the set. Table IV-2 contains an example of the measurable parameters and their corresponding sources. The complete set of measurable performance parameters and their corresponding sources are contained in Table E-7 of Appendix E.

Implementation Options

The evaluation of the capabilities of the existing performance tools presented several options for the actual implementation of a DBMS performance monitor, and these options were in the area of data analysis. For example, the monitor and accounting utilities have the capability to produce raw data files for user developed data analysis software, or they have the capability to produce ASCII formatted summary files which contain the required analysis. If the raw data files option is chosen, data analysis software must be provided or developed by the user. If the summary files option is chosen, the data is analyzed by the utility; however, the data is intermixed

TABLE IV-2

Examples of VAX and TOTAL Performance Parameters

Parameter Name	Source
System throughput	Accounting Utility
DBMS throughput	DBMS Log (DBMS Storage Functions only) or an Instrumented Program
System and DBMS turnaround time	Accounting Utility
System and DBMS response time	Instrumented Program
Component reliability	SYE Utility
CPU busy	Monitor Utility
DBMS CPU utilization	Calculated using Accounting Utility data
Number of page faults (system)	Calculated using Accounting Utility data
Number of page faults (DBMS)	Calculated using Accounting Utility data
I/O rates	Monitor Utility
Number of data base objects accessed	DBMS Log (DBMS Storage Functions only) or an Instrumented Program
Mean I/O statistics per DML statement	Calculated using data from an Instrumented Program
Device utilization (system)	SYE Utility
Mean length of system queues	Monitor Utility

with print headers generated by the utility. This does not present a problem if the VAX generated format is acceptable for the performance report, but if a different type of performance report is desired, the ASCII formatted summary files have to be read by a specialized program to locate and extract the values of the measurement data.

The data produced by instrumented applications programs also needs to be analyzed, and this analysis could be performed by either the instrumented program or separate data analysis software.

Additionally, two options existed in terms of data presentation. The data produced by each tool could be separately printed and consolidated by the analyst, or special presentation software could be written to consolidate the data and produce a consolidated performance measurement report.

Implementation Plan

For this study, it was decided to use the data analysis capabilities of the monitor and accounting utilities instead of developing new data analysis routines. Additionally, it was decided to create a consolidated performance measurement report, and this required the development of specialized software to extract the necessary performance parameter values from the ASCII formatted summary files. Typically, specialized software such as this should be avoided; however, developing new software for a function that already exists is too wasteful. Therefore, the analysis capabilities of the utilities and a specialized, well documented, data extraction program were chosen as the implementation method.

To make the instrumented program capability more powerful, it was decided that a generalized instrumentation utility would be developed. This allows any VAX user to instrument their applications programs through subroutine calls to the instrumentation utility. Also, it was decided the instrumentation utility would operate in two discrete steps. In the first step, it would only collect data and write it to a file, and after all the data was recorded, the second step would analyze the recorded data. Therefore, the instrumentation utility provides not only a general purpose utility to instrument programs, it can provide both generalized or specialized data analysis capabilities. This allows the instrumentation program to be more general purpose and be used for more than just DBMS oriented applications program.

To provide data analysis for the instrumentation utility, it was decided that data analysis software would be developed as a part of this study. This decision was made for two reasons. First, the level of data analysis required is relatively simple, and second, an online data analysis capability will provide better service than the alternative of creating data files for use on the CYBER computer.

A block diagram of the implementation plan is shown in Figure IV-4, and this figure shows how each major functional requirement is accomplished through a set of computer software. The blocks annotated with a (U) denote a VAX utility, while the blocks annotated with a (P) denote a program developed as a part of the study.

In terms of controlling the implementation, the user interface is used to generate a VAX command procedure, and this procedure contains all the necessary measurement control, analysis, and presentation commands in the form of VAX Command Language statements. These statements are used to initiate the necessary performance tools and user developed software comprising the DBMS performance monitor. After the user has completely specified the measurement session and selected the exit option, the user interface will display a message instructing the user to enter the following VAX command:

```
$SUBMIT/NOLOG_FILE [DBMON]DBMONINIT
```

This command submits the DBMS performance monitor as a

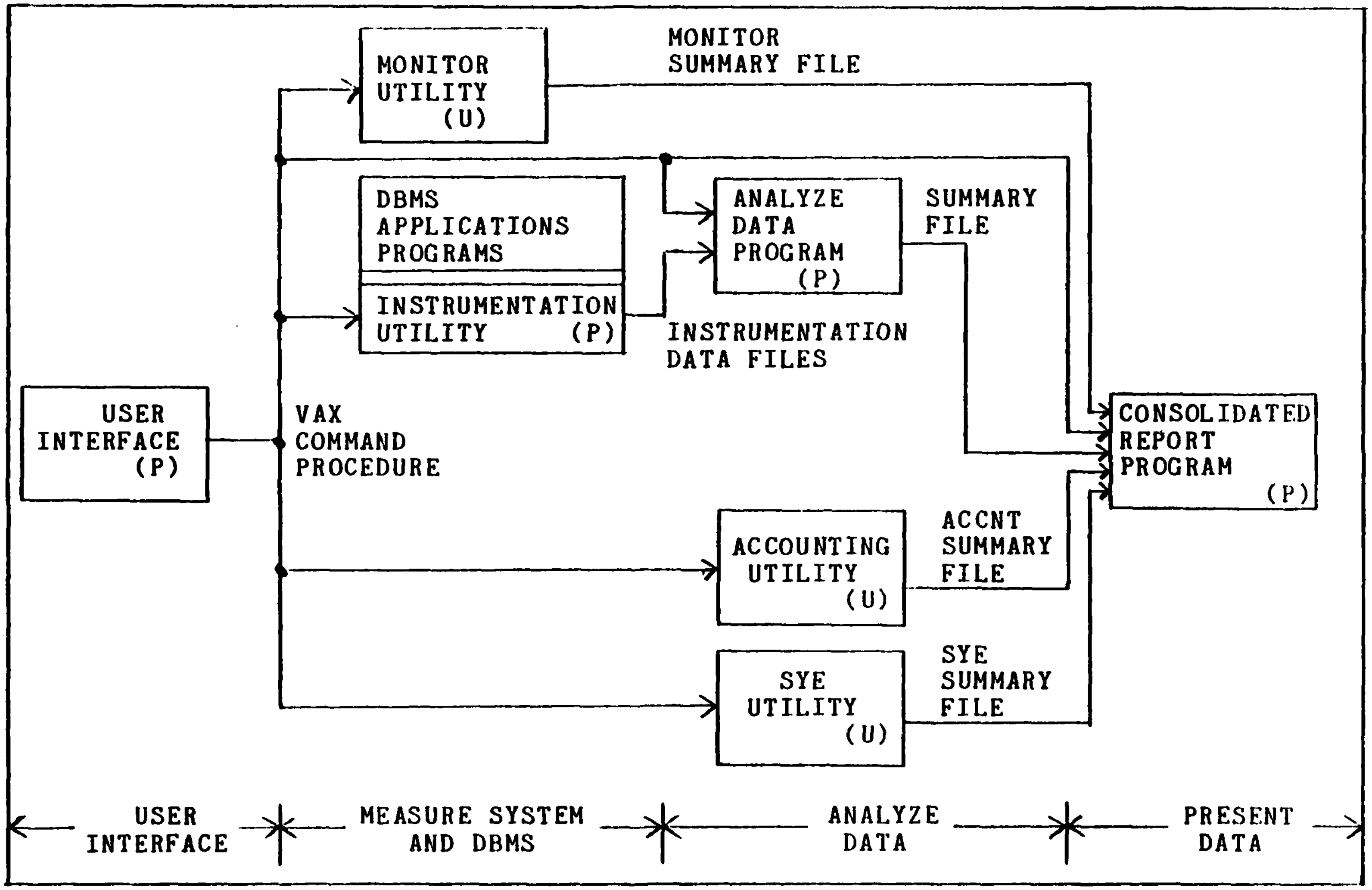


Figure IV-4. Implementation Plan

background batch job allowing the terminal to be used for other tasks.

Summary

This chapter examined the details of implementing the generalized design for a DBMS performance monitor on a VAX 11/780 computer and the TOTAL DBMS. First, the operational details of the VAX computer and TOTAL DBMS were presented to illustrate the importance of understanding the details of their operation. Next, the existing performance evaluation tools of the VAX computer and TOTAL DBMS were evaluated to determine the set of tools useful to the development of a DBMS performance monitor. Additionally, the VAX operating system was examined to determine if any "hooks" existed to facilitate the development of additional performance tools. Based on the results of the system evaluation, a plan for implementing the DBMS performance monitor was developed, and the next chapter discusses the details of actually implementing the DBMS performance monitor.

V. Program Design, Implementation, and Testing

Introduction

This chapter describes the development of the User Interface, Instrumentation Utility, Data Analysis, and Measurement Report programs identified in the implementation plan of Figure IV-4. Each program is described individually and the description includes: specialized data structures, high level program structure, and program testing along with results. In addition to the documentation presented in this chapter, Appendix F contains complete and detailed structure charts for each program.

Development Strategy

A development priority was assigned to each of the four programs, and the highest priorities were assigned to the programs responsible for data collection tasks. This strategy seemed reasonable since it is impossible to analyze or present data that has not been collected.

The user interface controls all the performance tools used in the DBMS performance monitor, and it was assigned the highest development priority. This allowed all existing VAX performance tools to begin collecting data as quickly as possible. Also, developing this program first provided two other advantages. It provided a basis for testing the generalized design presented in Chapter 3, and it provided a limited DBMS performance monitoring capability with the development of a single program.

The instrumentation utility is the only other program involving data collection tasks, and it was assigned second development priority. Of the two remaining programs, the data analysis program analyzes the data collected by the instrumentation utility, and the measurement report program produces a consolidated measurement report. Since the VAX performance tools can produce measurement reports (ref. 21:Chapter 1,12, and 17), the need for a consolidated measurement report is more of a "user friendly" feature than it is an essential requirement. Therefore, third priority for development was assigned to the data analysis program, and fourth priority for development was assigned to the measurement report program.

Testing Procedure

Before the test plan discussed in Chapter 3 can be used, a detailed procedure for applying the test plan must be developed. The procedure used in this study consists of four steps which are: unit testing, integration testing, validation testing, and system testing (ref. 41:295-305). As a whole, these steps are referred to as incremental testing because the testing proceeds on an incremental basis throughout the software development. An incremental testing process is consistent with the principles of software engineering (ref. 53:7-9,88-99 and 41:295), and if the testing is started with the design stage, it allows errors to be discovered as early as possible.

Each of the four testing steps are briefly described below:

<u>Unit Testing</u>. Unit testing focuses on testing each program module individually, and this step assures that each module functions correctly as an individual unit of a software system.

<u>Integration Testing</u>. After unit testing has been completed, the modules must be assembled or integrated to form a complete program. Integration testing focuses on testing the complete program to ensure the interfaces between modules are working correctly and that none of the modules have an inadvertent, adverse side effect on other modules.

<u>Validation Testing</u>. Validation testing is performed to ensure the assembled set of software functions in accordance with the requirements. Typically, a series of "black-box" test cases are used to perform the validation.

System Testing. System testing is the final step, and it is used to ensure all elements of the system work properly and that overall system function and performance requirements are achieved. This testing must involve the end-user of the software system, and it is typically culminated in a series of acceptance tests.

Programming Language Selection

Four languages were available for program development, and these four languages were MACRO-11 Assembly, FORTRAN, C, and PASCAL. Of these languages, PASCAL was chosen as the primary development language. PASCAL was chosen for several reasons, but the two primary considerations were PASCAL's ability to define data types and structures, and PASCAL is the language used at AFIT to develop applications programs for the TOTAL DBMS. (Other languages such as COBOL, FORTRAN, and MACRO-11 can also be used to develop TOTAL DBMS applications programs (ref. 8:1-4); however, none of these languages are used at AFIT).

The ability to define data types and structures is important to the process of developing programs. The first steps in program development concentrate on the data transformations performed by the program, and during these steps, a scheme for representing the data is devised (ref. 1:9-12; 30:4-5; and 53:89). After this has been accomplished, the algorithms operating on the devised data scheme are selected or developed. The end result of the initial development steps is a data structure consisting of the selected data types, the functions/algorithms operating on the data types, and the relationships between the functions/algorithms and the data types. In formal terminology, the data structure consists of a set of domains (data types), a set of functions (algorithms), and a set of axioms (relationships). This triple is referred to as an abstract data type, and its specification allows for a clear understanding of what the data structure is intended to do (ref. 1:11-14 and 30:7).

The next step in program development is to implement the data structure specification in a programming language. Ideally, the development language should contain constructs facilitating the implementation of the data structure specification, and PASCAL is such an language since it allows the user to define their own data types and structures.

Since AFIT applications programs for the TOTAL DBMS are developed in PASCAL, it made sense to select PASCAL as the development language. In this way, users of the DBMS performance monitor would already have some familiarity with the language if the monitor needed to be examined and/or modified.

Situations did occur during the program development stage where PASCAL could not be used because of limitations in its capability to interact with the VAX File System and System Services Library. Specifically, problems occurred during the development of the instrumentation utility. The details of the problems and the alternate programming language used are presented in the description of the instrumentation utility.

User Interface

The concept of iterative enhancement was applied to the user interface in the following way (ref. 53:54-55 and 4:121-127). The requirements analysis for a DBMS performance monitor showed the monitor must be flexible enough to allow users to select the set of performance parameters applicable to their "level of DBMS observation" and performance monitoring objectives. Before this capability can be developed, the ability to measure the entire set of all possible performance parameters should be demonstrated. Therefore, the initial development of the user interface only allows the user to select the option of measuring the enitre set of parameters. The ability to select pre-defined or specialized subsets should be a future expansion. Additionally, the data presentation options were limited to printed reports. The ability to display measurement reports on a terminal will also be a future expansion.

Data Structures. The three main features of a "user freindly" interface for the DBMS performance monitor were: allowing the user to navigate through the control paths of the interface, allowing the user to enter input data, and informing the user of errors. These features were provided through the use of menus, data entry prompts, and error messages, and the PASCAL data type statement was used to define descriptive names for the individual menus, prompts, and messages. In this way, an easy mechanism for specifying individual data objects to the functions operating on the defined names was developed, and the functions were easily implemented through the use of case statements. An example of the type definition for the menu names and the DISPLAYMENU procedure is shown in Figure V-1.

Program Structure. The high level structure of the user interface is shown in Figure V-2. The user interface was designed using a state diagram to determine the control paths through the user interface. After this was completed, each state was evaluated for control path options, data entry requirements, and possible error processing/messages. Using the results of the evaluation, the last design step was to develop a set of conventions for the menus, data entry prompts, and error messages. A set of conventions is necessary because it assures the

```
TYPE
   MENUNAME = (MAIN,PARAMETERS,ANALYSIS,.....);
        .
        .

PROCEDURE DISPLAYMENU(NAME : MENUNAME);
BEGIN
   CASE NAME OF

      MAIN:  (* PASCAL CODE TO DISPLAY THE MAIN MENU *)

      PARAMETERS:  (* PASCAL CODE TO DISPLAY PARAMETERS
                      MENU *)
           .
           .
      END     (* END CASE *)
END;  (* END PROCEDURE DISPLAYMENU *)
```

Figure V-1. Example of Data Type Definition

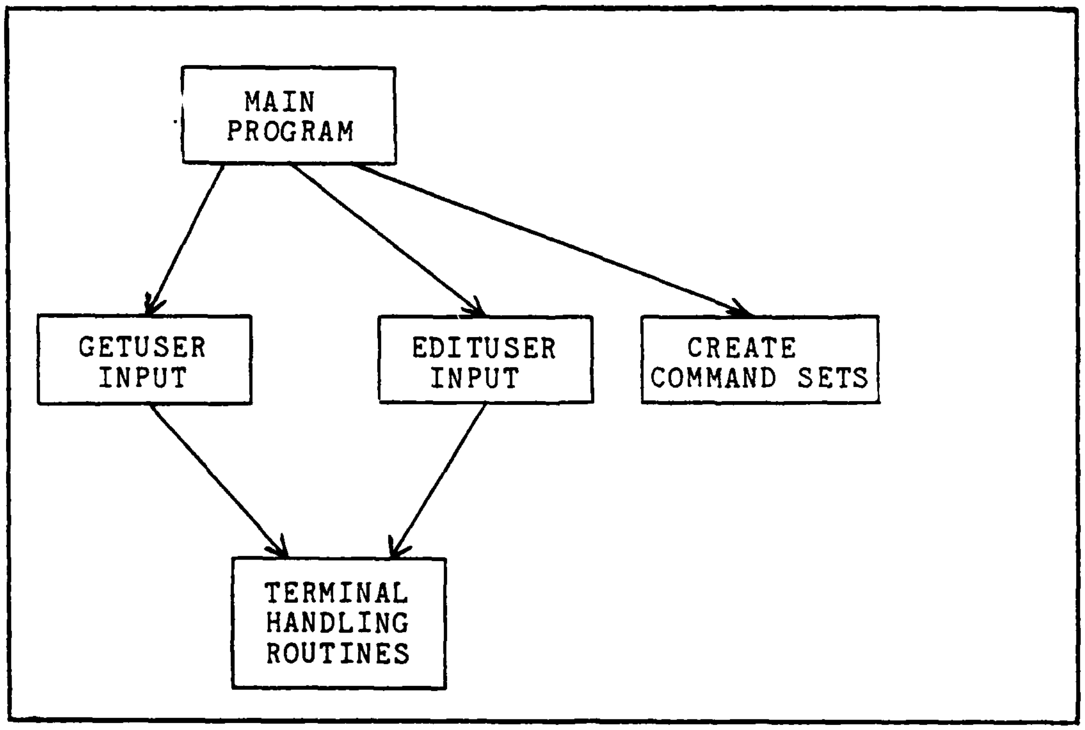


Figure V-2. User Interface Structure

operation of the program remains consisent, thereby, reducing the potential for human error (ref. 38:257 and 44:112-114).

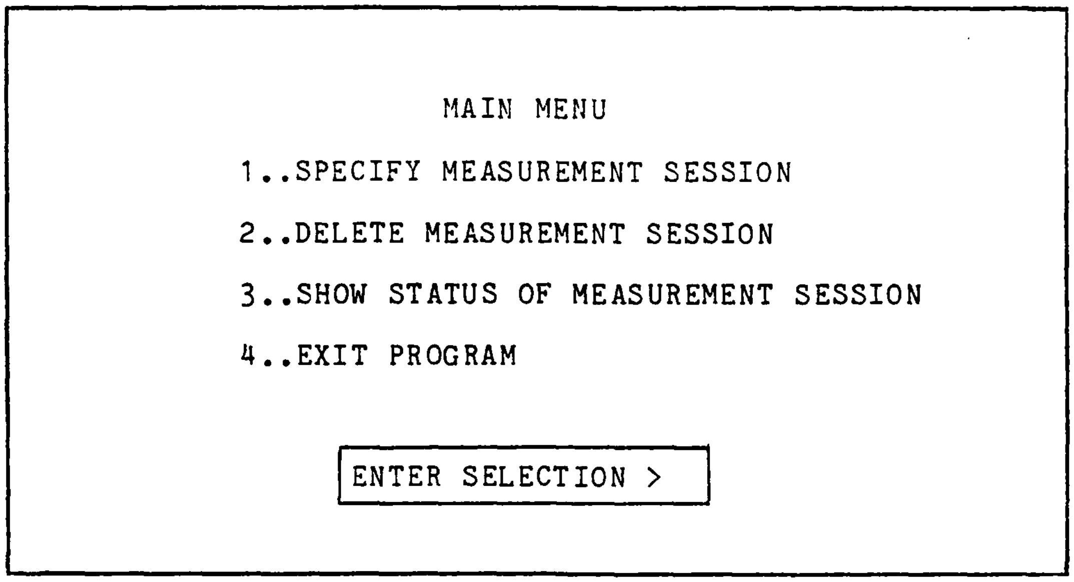
The state diagram evaluation showed two types of data needed to be entered by the user. The first type is static data which maintains a constant definition for all executions of the DBMS performance monitor. Examples of static data are pre-defined performance parameter subsets, functions performed by the user interface, data analysis options, etc.. Menus with descriptive titles were used to enter static data, and the user makes the desired selection by entering a single number. An example menu is shown in Figure V-3, and the conventions established for the menus are:

1. The menu title is always displayed in double height characters, centered in lines 2 and 3 of the display.

2. Menu selections are always numbers.

3. The first menu selection line always starts on line 6, and the last possible menu selection line is line 16.

4. The enter selection prompt is always displayed in line 18 in reverse video.



```
                MAIN MENU
   1..SPECIFY MEASUREMENT SESSION
   2..DELETE MEASUREMENT SESSION
   3..SHOW STATUS OF MEASUREMENT SESSION
   4..EXIT PROGRAM

          ENTER SELECTION >
```

Figure V-3. Example Menu

The second type of data is dynamic data, and this type of data can vary for each execution of the DBMS performance monitor. Examples of dynamic data are: the start date, the stop date, the stop time, etc.. Full prompting, with examples, was used for the entering of dyanamic data. An example of a data entry prompt is shown in Figure V-4, and the following conventions were established for data entry prompts:

1. Information and/or examples relating to data entry prompts always begin in line 12.

2. The user prompt line is always displayed in reverse video.

```
ENTER STOP TIME FOR MONITOR SESSION
FORMAT IS HH:MM:SS, EXAMPLE 18:45:00
PRESS <RETURN> FOR DEFAULT STOP TIME OF 24:00:00

ENTER STOP TIME >        < END-OF-INPUT-FIELD
```

Figure V-4. Example Data Entry Prompt

3. The length of the data entry field is shown.

In addition to allowing the user to enter data, the data was always edited as fully as possible to detect and correct errors at the earliest possible time. This capability required the display of error messages when errors were detected. Error messages are always as informational as possible, and to help to user to recognize the mistake, the original entry is not erased. The cursor is re-positioned to the beginning of the data entry field, and the user is allowed to type over the previous entry. The following convention was established for the display of error messages:

Error messages are always displayed in lines 23 and 24 of the display in double height characters.

A special set of terminal handling routines were developed for the user interface to help make it as "user friendly" as possible. These routines performed functions such as: positioning the cursor to a specified row and column number, erasing parts of or the entire display, drawing boxes around items, displaying items in reverse video, etc.. The capabilities provided by these routines enhanced the appearance of the screen formats and provided a more pleasing environment than scrolling. These routines were established and maintained in a separate external library. This allows the user interface to be re-hosted on a different type of terminal by simply modifying the external terminal routines. The user interface program remains independent of the terminal type.

After a measurement session is completely specified, the data entered by the user is always displayed for review. If changes need to be made, individual data items can be selected for modification. All modifications are fully prompted in addition to displaying the current value. After all the modifications have been made, the data is re-displayed. If the data is correct, the command sets for controlling the DBMS performance monitor are created; otherwise, the user can continue to review and modify the data. Additionally, the data is written to a session log file for use by other programs. The information recorded in the log record is shown in Table V-1.

<u>Testing and Results</u>. The user interface was tested in accordance with the test plan developed during the system design stage (ref. III-10 and Appendix D). For those features that have been implemented, the user interface program has successfully passed all tests conducted during the unit, integration, and validation testing steps. No design modifications were required.

Instrumentation Utility

The instrumentation utility is the mechanism used to obtain detailed performance information at the DML statement level of a DBMS. Ideally, this utility should be designed as a module of the DBMS, allowing individual users to enable or disable its operation. However, the TOTAL DBMS does not have this utility designed within it, and it must be provided in some other way. Two approaches to developing the instrumentation utility were considered. The first approach involved modifying the TOTAL DBMS to include this utility, but as stated in the system analysis chapter, this approach entails a high deal of risk. It could potentially introduce errors into the DBMS, and it makes it difficult to maintain DBMS updates. Additionally, this approach requires an intimate knowledge of the

Table V-1

Measurement Session Log Information

Name of the Measurement Session
Start Date
Start Time
Stop Date
Stop Time
Data Analysis Options
Data Presentation Options
Performance Parameter Set
Data Collection interval for the VAX/VMS Monitor Utility
Status of the Measurement Session

software modules comprising the TOTAL DBMS, and this information was not available.

The second approach, the one used in this study, is to provide users with an instrumentation utility that can be incorporated into applications programs. The VAX 11/780 System Services Library provided the necessary system routines for accessing performance data maintained by the operating system tables (ref. 18:124-132), and a set of external procedures (subroutines) were developed to allow easy access to the capabilities provided by the system services library.

Three external procedures were defined as the interface to the instrumentation utility. To instrument a DBMS applications program, a user defines the three external procedures in the PASCAL applications program, and calls to the procedures are included in the applications program to allow performance parameter values to be measured and accumulated. The names of the three procedures and the functions they perform are described below, and Figure V-5 illustrates how the instrumentation utiltity is incorporated into an applications program.

1. INITUTILITY - This procedure initializes the instrumentation utility and creates a data file for accumulating performance parameter values. This procedure is typically called once before the data base signon request is sent to the TOTAL DBMS, and the other two procedures are called on a DML by DML statement basis. However, the capability exists to call this procedure more than once, thereby, creating more than one data file. This capability gives the user the flexibility to tailor the contents of the data file for use in specially designed performance tests. Two parameters, the program and data base names, are inputs to this procedure, and a completion status is the output from the procedure. The two input parameters are PASCAL string types, and they can be represented either as a literal or variable name. Since the status is supplied by the external procedures, a variable name must be specified for the output parameter.

2. MEASUREDBMS - This procedure records the state of the DBMS just prior to the execution of a DML statement (or group of DML statements), and a call to this procedure should immediately precede a DML statement. In

```
PROGRAM COURSEDATA(INPUT,OUTPUT);

TYPE
   BUFF4  = PACKED ARRAY [1..4] OF CHAR;
   BUFF5  = PACKED ARRAY [1..5] OF CHAR;
   BUFF6  = PACKED ARRAY [1..6] OF CHAR;
   BUFF15 = PACKED ARRAY [1..15] OF CHAR;
      .
      .
PROCEDURE INITUTILITY(%STDESCR PROGRAMNAME : BUFF15;
                      %STDESCR DATABASENAME : BUFF6;
                      %STDESCR STATUS : BUFF4);
                      EXTERN;

PROCEDURE MEASUREDBMS(%STDESCR DBMSFUNCTION : BUFF5;
                      %STDESCR DBMSFILENAME : BUFF4;
                      %STDESCR STATUS : BUFF4);
                      EXTERN;

PROCEDURE ENDMEASURE(%STDESCR ENDCODE : BUFF6;
                     %STDESCR STATUS : BUFF4);
                     EXTERN;
      .
      .
PROCEDURE GETCOURSEDATA;

          PROCEDURE DATBAS(Call Parameters...); EXTERN;

     BEGIN
           .
        COMMAND := 'READM';
        FNAME := 'CRSE';
           .
        MEASUREDBMS(COMMAND,FNAME,STATCODE);
        DATBAS(Parameters comprising a DML statement);
        ENDMEASURE('ENDDML',STATCODE);
           .
     END;  (* END PROCEDURE GETCOURSEDATA *)

BEGIN  (* BEGIN MAIN PROGRAM *)
      .
   INITUTILITY('UPDATECOURSES  ','AFITDB',STATCODE);
      .
   GETCOURSEDATA;
      .
 END.
```

Figure V-5. Using the Instrumentation Utility

the case of the TOTAL DBMS, a call to this procedure immediately precedes a call to the DATBAS subroutine. Two parameters, the name of the DBMS function (DML command) and the name of the data base file, are inputs to this procedure, and a completion status is the output from the procedure. The two input parameters are PASCAL string types, and they can be represented either as a literal or variable name. Since the status is supplied by the external procedures, a variable name must be specified for the output parameter.

3. ENDMEASURE - This procedure records the state of the DBMS just after the execution of a DML statement (or group of DML statements), and a call to this procedure should immediately follow a DML statement. In the case of the TOTAL DBMS, a call to this procedure should immediately follow a call to the DATBAS subroutine. One parameter, the endcode, is the input to this procedure, and a completion status is the output from the procedure. The input parameter is a PASCAL string type, and it can be represented either as a literal or a variable name. Since the status is supplied by the external procedures, a variable name must be specified for the output parameter. Only two values are defined for the endcode. The first value is 'ENDDML' and this code denotes the end of a DML or group of DML statements, allowing additional calls to be made to the instrumentation

utility. The second value is 'ENDPRG', and this code denotes the end of a measurement session. It causes the data file to be closed, and the instrumentation utility is disabled. Therefore, the procedure INITUTILITY must be called to re-enable it.

Two status codes are returned by the procedures. A successful completion is indicated by the value '****', and this is consistent with the successful completion code of the TOTAL DBMS (ref. 8:2-23). An error condition is indicated by the value 'EROR', and this value was selected because it is a four character code with a close resemblance to the word error. In addition to returning the error condition, the instrumentation utility is disabled. Therefore, additional calls to procedures MEASUREDBMS and ENDMEASURE will have no effect until the instrumentation utility is re-initialized by calling the procedure INITUTILITY.

Data Structures. One special data structure was required by the instrumentation utility, and this structure was a requirement of the SYS$GETJPI procedure in the VAX System Services Library (ref. 18:124-126). This data structure is a list of item descriptors for the performance parameter values retreived from the operating system tables. The format of an element of this data structure is illustrated in Figure V-6.

To record the measured values of the performance parameters, a data record was created and written to a file. The contents of this record are shown in Table V-2.

Program Structure. The high level structure of the instrumentation utility is shown in Figure V-7. This diagram shows the three entry points into the instrumentation utility. This represents the external call structure from an applications program to the three procedures making up the instrumentation utility.

Testing and Results. The development of this program was initially attempted in PASCAL. Small modules were successfully developed and individually tested, but these modules could not be successfully integrated into an instrumentation utility and still remain easy to use. The major problems were:

1. An external PASCAL module cannot perform file operations without the file being duplicately and exactly defined in the calling program.

2. The itemlist descriptor elements could not be reliably allocated. Occasionally, the System Service

```
*********************************************
*      length       *       item code       *
*********************************************
*    address of buffer to receive data      *
*********************************************
*  address of buffer to receive data length *
*********************************************
```

Figure V-6. Item List Descriptor

Table V-2.

<u>Measurement Data Record</u>

Relative Record Number
Completion Code
Program Name
Data Base Name
Data Base File Name
Data Base Function
Before Call Date/Time
Before Call CPU Time
Before Call Buffered I/O Count
Before Call Direct I/O Count
Before Call Page Fault Count
Before Call Working Set Size
After Call Date/Time
After Call CPU Time
After Call Buffered I/O Count
After Call Direct I/O Count
After Call Page Fault Count
After Call Working Set Size

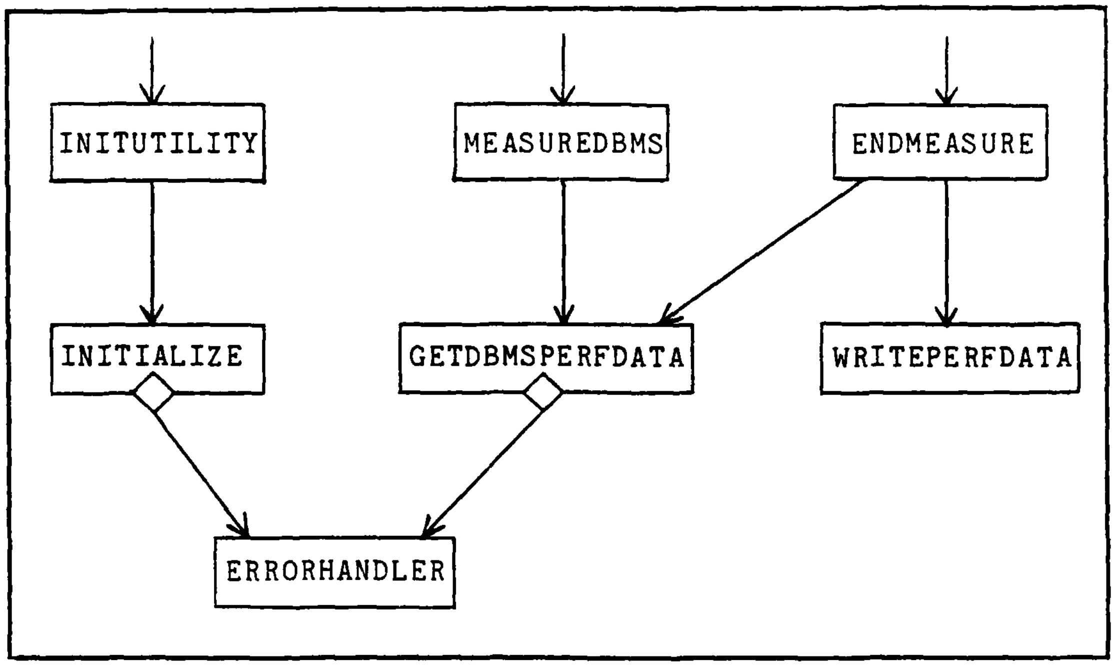


Figure V-7. Instrumentation Utility Structure

Library would return alignment errors meaning it expected the storage location to begin on a longword boundary which it did not.

3. Pointers were used to allocate the address values required in the definition of each item list descriptor element. However, the amount of new storage that can be allocated by the NEW() function is limited in external PASCAL modules.

In an attempt to solve these problems, a different approach was tried. Instead of the instrumentation utility being a library of subroutines, it was implemented as a stand-alone program. However, this meant the interface between applications programs and the instrumentation utility could no longer be simple subroutine calls. The new interface required the capability for two independent processes to communicate, and the VAX Mail Utility was used to provide this service. Using the Mail Utility, the new interface consisted of Mail messages transmitted between the two processes. Initially, this approach seemed to be a viable solution; however, two-way communications proved to be unreliable. The main problem occurred when two or more applications programs attempted to access the instrumentation utility simultaneously. PASCAL treats the mailbox messages as records read from a sequential file (ref. 19:6-6), and as soon as one of the programs finished using the instrumentation utility and closed the mailbox file, all remaining programs also lost their communications capability through the mailbox.

At this point it was decided to change the development language to MACRO-11 assembly. This language allows the most flexibility and capability to access the VAX File System and System Services Library. Therefore, the problems encountered using PASCAL were solved by using the assembly language capabilites, and this allowed the simple interface mechanism of calls to external procedures (ref. Figure V-5) to become feasible again. The MACRO-11 procedures are operational, and they have successfully completed unit, integration, and validation testing. These procedures are successfully accessing the system tables and returning the required performance data.

Data Analysis Program

This program analyzes (reduces) the data collected by the instrumentation utility. Before this program is executed, individual data files produced by the instrumentation utility are merged into one large file using the COPY command of the VMS operating system. Records from this file are individually retreived and processed to compute performance parameter values for applications programs and the DML statements used in the programs.

Data Structures. To compute total and average values for the performance parameters, a data structure is needed to accumulate the individual values read from the raw data file. Tables (arrays) were chosen as the data structures used to accumulate the values, and a hashing function was chosen as the method used to access specific loactions in the table. The hash value for a table key is computed by summing the integer value of each character in the key, dividing by the table size, and using the remainder as the hash value. To handle collisions, the technique of linear probing was used (ref. 1:126 and 30:464).

A simple linear (sequential) search could have been used to access specific table locations (ref. 30:335-336); however, the data analysis program needs to maintain more than one table. Specifically, separate tables for the DML commands, data base names, and data base file names are used. Since a linear search algorithm has an average performance of searching half of the table for each access, the computing time of the search algorithm can become significant as the size of the table increases (ref. 30:336). The computing time of hashing techniques is typically better than linear search techniques (ref. 30:469); however, the computing time of a hashing function can approach the computing time of a linear search if careful attention is not paid to the loading density of the table (ref. 30:469).

To compare the two techniques, Table V-3 shows the average number of searches required to search a table with 30 entries. This example assumes the table has a loading density of .8 (e.g. 24 locations in the table contain vaild data and the other six locations are empty). If three tables are used to accumulate values and each table is accessed once for each record read from the raw data file, linear searching requires an average of 46.5 searches while the hashing technique requires an average of only nine searches, a 500 percent improvement.

Table V-3

<u>Average Number of Searches</u>

Algorithm	Average Number of Searches Formula	Average Number of Searches
Linear Search	$(n + 1)/2$	15.5
Hashing (with linear probing)	$.5(1 + (1/1 - ld))$	3.0

n = 30
ld = loading density = .8 (24/30)

Formulas obtained from ref. 30:336,470

To implement the hashing technique, five routines were developed, and each of them are briefly described:

1. HASH - Computes the hash value of the key used to access the table.

2. NEXTPROBE - Computes the next probe point into the table when a collision occurs.

3. INITABLE - Initializes all the hash table locations to empty.

4. INSERT - Inserts items into the hash table.

5. LOOKUP - Finds the location of an item within the hash table.

Before the data analysis program is executed, the individual data files generated by the instrumentation utility are merged into one file. Typically, the data in the merged file is composed of data from many different programs; however, the data from individual programs is not intermixed within the file. Data from one program appears as a series of sequential data records. Therefore, the data from each program can be processed serially until data from a new program or end-of-file is encountered. The last record of each data file generated by the instrumentation utility should contain a last record indicator in the relative record number field of the file. However, situations could occur where the indicator is not generated because the users program did not terminate correctly due to programming logic errors, system failures, DBMS errors, etc.. Therefore, a control mechanism was needed to keep the data analysis program as robust as possible while reading the data file, thereby, preventing data from two separate programs from being accidently merged. The control mechanism was provided by a simple integer flag used to control the reading of records from the raw data file.

Program Structure. The high level structure of the data analysis program is shown in Figure V-8.

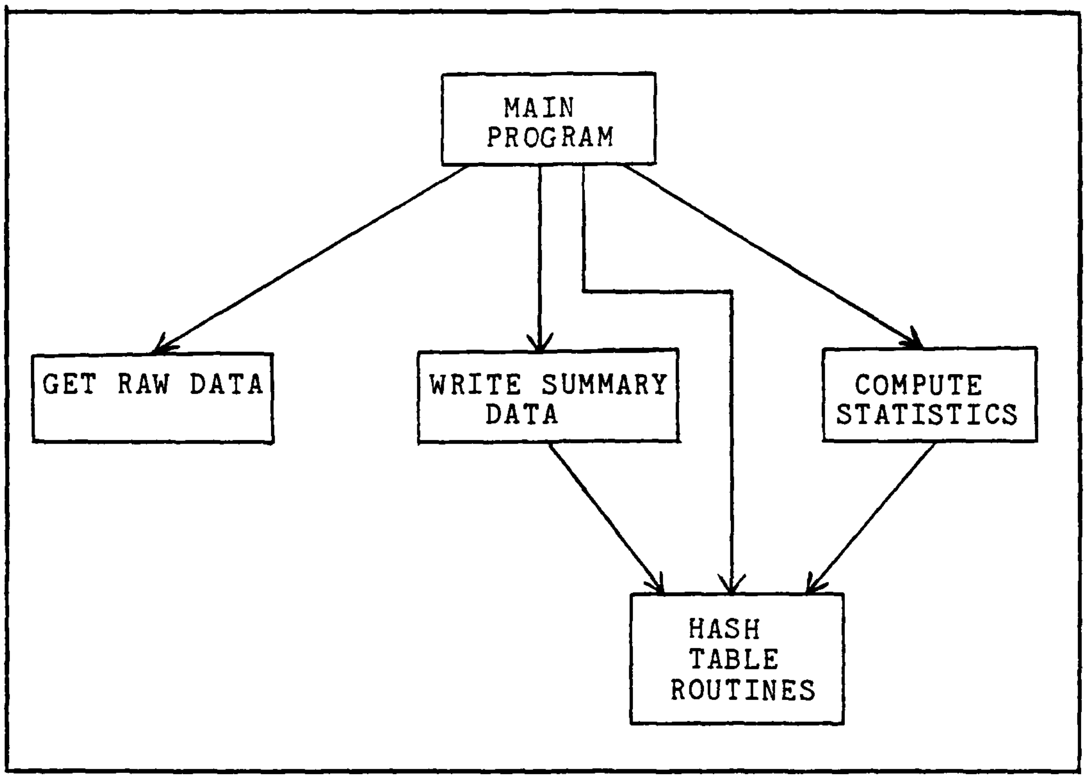
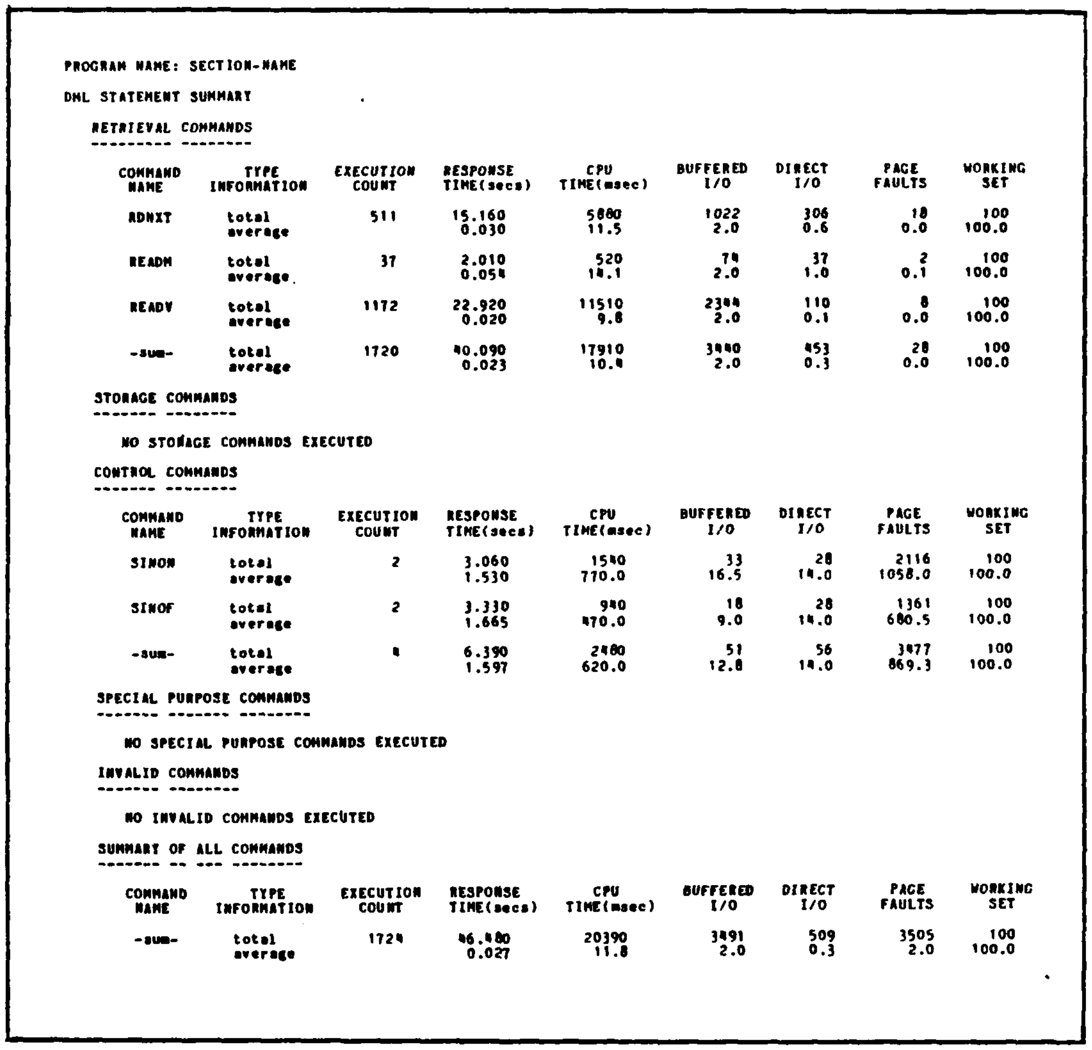


Figure V-8. Data Analysis Program Structure

Testing and Results. This program is operational; however, its data analysis capabilities are limited to only accumulating totals and computing average values for the DML commands. It has successfully completed unit, integration, and validation testing. A sample of the output generated by this program is shown in Figure V-9.

Measurement Report Program

This program consolidates the summary files produced by the VAX utilities and the data analysis programs. The generalized formats of the VAX summary files are not well



```
PROGRAM NAME: SECTION-NAME

DML STATEMENT SUMMARY

   RETRIEVAL COMMANDS
   --------- --------

     COMMAND      TYPE       EXECUTION   RESPONSE      CPU      BUFFERED  DIRECT     PAGE    WORKING
      NAME     INFORMATION     COUNT    TIME(secs)  TIME(msec)     I/O      I/O     FAULTS      SET

     RDNXT       total          511       15.160       5880       1022      306        18      100
                 average                   0.030       11.5        2.0      0.6       0.0    100.0

     READM       total           37        2.010        520         74       37         2      100
                 average                   0.054       14.1        2.0      1.0       0.1    100.0

     READV       total         1172       22.920      11510       2344      110         8      100
                 average                   0.020        9.8        2.0      0.1       0.0    100.0

     -sum-       total         1720       40.090      17910       3440      453        28      100
                 average                   0.023       10.4        2.0      0.3       0.0    100.0

   STORAGE COMMANDS
   ------- --------

     NO STORAGE COMMANDS EXECUTED

   CONTROL COMMANDS
   ------- --------

     COMMAND      TYPE       EXECUTION   RESPONSE      CPU      BUFFERED  DIRECT     PAGE    WORKING
      NAME     INFORMATION     COUNT    TIME(secs)  TIME(msec)     I/O      I/O     FAULTS      SET

     SINON       total            2        3.060       1540         33       28      2116      100
                 average                   1.530      770.0       16.5     14.0    1058.0    100.0

     SINOF       total            2        3.330        940         18       28      1361      100
                 average                   1.665      470.0        9.0     14.0     680.5    100.0

     -sum-       total            4        6.390       2480         51       56      3477      100
                 average                   1.597      620.0       12.8     14.0     869.3    100.0

   SPECIAL PURPOSE COMMANDS
   ------- ------- --------

     NO SPECIAL PURPOSE COMMANDS EXECUTED

   INVALID COMMANDS
   ------- --------

     NO INVALID COMMANDS EXECUTED

   SUMMARY OF ALL COMMANDS
   ------- -- --- --------

     COMMAND      TYPE       EXECUTION   RESPONSE      CPU      BUFFERED  DIRECT     PAGE    WORKING
      NAME     INFORMATION     COUNT    TIME(secs)  TIME(msec)     I/O      I/O     FAULTS      SET

     -sum-       total         1724       46.480      20390       3491      509      3505      100
                 average                   0.027       11.8        2.0      0.3       2.0    100.0
```

Figure V-9. Sample Output from Data Analysis Program

related to all aspects of DBMS performance. For example, the summary files generated by the Accounting Utility do not directly relate resource consumption by the TOTAL DBMS to resource consumption by all other programs (ref. 21:Chapter 1). This type of information is necessary to determine relationships such as: the percentage of CPU time used by the DBMS, the percentage of page faults attributable to the DBMS, the percentage of I/O activity attributable to the DBMS, etc.. Therefore, this program reads the individual VAX summary files and extracts the necessary values for DBMS performance parameters. After the values have been extracted, a performance measurement report is created in a format relating the extracted performance parameter values to the different aspects of computer system and DBMS performance.

Testing and Results. Due to time limitations and problems with developing the instrumentation utility, program implementation was not initiated.

Summary

This chapter presented the development of the four programs required by the implementation plan of Chapter 4. The high level structure of each program was presented in addition to any special data structures used by the program. The testing results of each program were presented, and the only major problem occurred in the development of the instrumentation utility. This problem was attributable to programming language incompatibilty, and its effect was to delay the schedule such that the measurement report pr[illegible]ram was not developed.

The next chapter presents the results of testing the

DBMS performance monitor as a complete system. Additionally, the results of using the monitor to measure a VAX 11/780 computer and TOTAL DBMS are presented.

VI. Results, Conclusions, and Recommendations

Introduction

This chapter presents the results of testing the DBMS performance monitor as an operational system and the results of using the monitor to measure the TOTAL DBMS operating on a VAX 11/780 computer. Additionally, the conclusions and recommendations derived from the results of this study are presented.

System Testing

With the exception of the measurement report program, the DBMS performance monitor is operational and capable of measuring DBMS performance in a VAX 11/780 environment. No problems were discovered with the generalized design presented in Chapter 3, and during the development and testing process, the only significant problem was the incompatibility of the PASCAL language for use in developing the instrumentation utility.

The lack of a measurement report program for consolidating the summary reports generated by the individual performance tools is a drawback. It makes the interpretation of the output cumbersome because it has to be manually interpreted. However, all the necessary information is available.

Using the DBMS Performance Monitor

This section describes the operational environment of the VAX 11/780 computer and the TOTAL DBMS. Since most of the work in this study was dedicated to developing the monitor, only a limited amount of measurements were taken. These measurements provide a general idea of DBMS performance; however, more measurements need to be taken with the instrumentation utility to obtain a better understanding of DBMS performance at the DML statement level. For example, the following types of performance tests need to be conducted: test the effects of increasing the number of simultaneous users, test the effects of increasing the data base size, test the effects of changing data base generation parameters, test the effects of file linkpaths, etc..

VAX 11/780 Operational Environment.

The DBMS performance monitor was used on the VAX 11/780 computer described in Chapter 4. Under the current configuration (ref. Figure IV-1), disk drive 0 is dedicated to the VMS operating system, and disk drive 1 is available for user disk volumes. Since the storage space on a disk volume is limited to 28 megabytes, enough space is not available on a single volume to satisfy the requirements of all users. Specifically, the TOTAL DBMS resides on a disk volume different from the normal user disk volume.

The workload for this computer is primarily generated by student research; however, it does perform a limited amount of production work for the school administration. On any given day, the wor''ↄad for this computer varies greatly, and it is hypothesized that the workload is non-stationary. However, no statistical tools were easily available to verify this hypothesis.

DBMS Operational Environment. Currently, the TOTAL DBMS is being used for two purposes. The first purpose is to maintain a production data base for use by the EE Department, and the second purpose is to maintain a development environment for students in the Computer Data Base Systems course. Because of the problems with limited mass-storage space, the TOTAL DBMS is never simultaneously used to support both purposes. Therefore, the TOTAL DBMS is used in two different modes. The first mode is a stand-alone mode for production work, and the second mode is a multi-user mode for development work on student data base projects.

This environment created a situation where the system had little activity during the measurement sessions. Typically, the system activity was limited to the TOTAL DBMS, a single DBMS applications program, and one or two interactive terminals performing program development tasks such as: text editing, compiling, and test executions. Under these conditions, the VAX computer showed less than 50 percent CPU utilization where the utilization was evenly divided between the VMS operating system and the users. Also, over half of the available main memory was not being used. Therefore, the measurement results presented in the next section must be weighed against the relative inactivity of the system during the period of time it was being measured.

Monitor Operation. The details of using the DBMS Performance Monitor are not presented in this chapter; however, a complete Users Manual is contained in Appendix G.

Measurement Results

This section presents the results of using the DBMS performance monitor, and these results fall into two categories. The first category is a high level view of DBMS performance, and the second category is a detailed view of DBMS performance at the DML statement level.

High Level View. The measurement results presented for this level were primarily obtained by the VAX Monitor and Accounting Utilities, and these measurements are presented by specific resource.

CPU - Approximately 10 to 15 percent of the available CPU time was consumed by the TOTAL DBMS.

Memory - Because of the large amount of free memory space, less than 6 percent of the page faults generated by the TOTAL DBMS required actual disk reads. The remainder of the page faults were serviced by accessing the free or modified page lists.

I/O - The number of direct I/Os the TOTAL DBMS issues to the disk is dependent upon the logic of the DBMS applications program; however, the TOTAL DBMS was consistently the largest user of the disk resources. In some cases it accounted for almost 45 percent of the total number of direct I/Os issued by all jobs executed during the measurement session. However, this did not present a performance bottleneck since the utilization of the I/O subsystem was calculated to be less the 5 percent.

From these measurements, it was concluded the TOTAL DBMS is currently enjoying a performance advantage due to the abundance of main memory and relative inactivity of the VAX computer. Response times remain fast as there is little competition for the processor and I/O subsystem.

<u>Detailed View</u>. The instrumentation utility was used to conduct three performance tests at the DML statement

level. For each of the tests, the TOTAL DBMS was being accessed by a single user and only one data base was being accessed. The data base consisted of five master files and two variable files where multiple linkpaths exist between master and variable files.

The three performance tests are described in detail below:

Test One. The goal of this test was to provide a general idea of DBMS performance. Table VI-1 shows the results of this test, and the values in the table are grouped by the four general categories of DML statements. However, not all of the possible DML commands were measured. The measurements presented in Table VI-1 were made using three retrieval commands (RDNXT, READM, and READV), three storage commands (ADD-M, ADDVB, and DELVD), and two control commands (SIGNON and SIGNOF). No special purpose commands were measured.

The values in the table are average values; however, they should not be considered typical values for all applications. These values will vary based on: data base generation parameters, data base size, number of files within the data base, linkages between the data base files, the number of simultaneous users of TOTAL (and the VAX 11/780), the VAX 11/780 configuration, and the VAX 11/780 parameters assigned to the account

Table VI-1

Performance Results at the DML Statement Level

COMMAND CATEGORY	RESP TIME (sec)	CPU TIME (msec)	DIRECT I/O COUNT	BUFFERED I/O COUNT	PAGE FAULT COUNT
RETREIVAL	.024	10	1	2	0.0
STORAGE	.070	17	2	2	0.0
CONTROL	1.403	230	14	13	8.0
SPECIAL	<Not Measured>				

executing the TOTAL DBMS (e.g. working set quota, priority, etc.). Even though these values can not be considered typical, they do provide some insight into DBMS performance. Specifically, the control category of DML commands have performance values an order of magnitude greater than the other two categories, retrieval commands have the best performance values, and the overall response time is composed of approximately one-third CPU time and two-thirds I/O and other wait times.

Test Two. The goal of this test was to determine the relationship of available main memory and working set size to DBMS performance. For this test, a DBMS applications program using a standardized set of DML commands was executed against various memory configurations. The set of DML commands consisted of 1,720 retrieval commands (511 RDNXT; 37 READM; and 1,172 READV) and four control commands (2 SIGNON and 2 SIGNOF). The results of this test are contained in Table VI-2, and the following conclusions were drawn from this test:

1. DBMS performance begins to stabilize at a working set size of 80 pages. Additional increases in working set size had little effect on performance except for control commands. The control commands generate a large number of page faults and require more CPU time. However, the control commands are executed very infrequently in a TOTAL DBMS applications program. Hence, increasing working set size above the stabilization point for the commonly used DML commands would not be cost effective. Based on this, a working set size of 80 pages is all this is required to obtain satisfactory performance; anything above this value will add little to DBMS performance!

2. A working set size less than 80 pages is not recommended. The response time values shown in Table VI-2 for working set size less than 80 pages are artificially fast at main memory sizes of .5 and .75 megabytes because there was little other activity on the measured system. At a main memory size of .5 megabytes and working set size of 50, a total of 92,705 page faults were generated, but only 400 of these required page reads from the disk. All other page faults were serviced by the free

AD-A149 950   DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM   3/3
PERFORMANCE MONITOR VOLUME 1(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..   P D BAILOR
UNCLASSIFIED   DEC 83 AFIT/GCS/EE/83D-2-VOL-1   F/G 9/2   NL

END
FILMED
DTIC

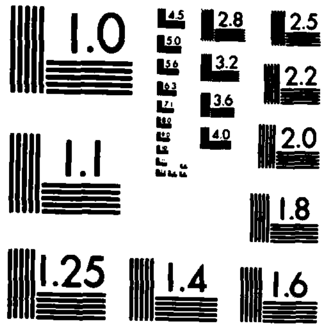

1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Table VI-2

Results of Main Memory and Working Set Test

			AVERAGES PER DML COMMAND			PAGE FAULT TOTALS FOR ALL COMMANDS	
MAIN MEMORY SIZE	WORKING SET SIZE	DML COMMAND CATEGORY	RESPONSE TIME (secs)	CPU TIME (msecs)	NUMBER PAGE FAULTS	TOTAL PAGE FAULTS	PAGE FAULT READS
.5	50	Retrieval	.046	29	50.0	92,705	400
		Control	4.957	1057	1,669.0		
.75	50	Retrieval	.045	30	49.8	92,319	212
		Control	2.537	972	1,655.2		
.75	60	Retrieval	.038	25	37.4	69,710	59
		Control	2.178	807	1,349.8		
.75	70	Retrieval	.027	13	6.7	16,172	61
		Control	1.945	712	1,166.3		
.75	80	Retrieval	.023	10	0.1	4,333	123
		Control	2.725	752	1,058.3		
.75	90	Retrieval	.023	10	0.0	4,325	108
		Control	2.730	695	1,035.8		
.75	100	Retrieval	.023	10	0.0	3,464	60
		Control	1.765	607	861.8		

Main Memory Size is specified in Megabytes
Working Set Size is specified in number of pages

Table VI-2 (Continued)

Results of Main Memory and Working Set Test

			AVERAGES PER DML COMMAND			PAGE FAULT TOTALS FOR ALL COMMANDS	
MAIN MEMORY SIZE	WORKING SET SIZE	DML COMMAND CATEGORY	RESPONSE TIME (secs)	CPU TIME (msecs)	NUMBER PAGE FAULTS	TOTAL PAGE FAULTS	PAGE FAULT READS
.75	200	Retrieval	.023	10	0.0	172	29
		Control	1.465	292	42.3		
2.5	50	Retrieval	.045	30	49.9	92,481	14
		Control	1.872	950	1,665.0		
2.5	60	Retrieval	.040	26	37.2	70,303	14
		Control	1.795	835	1,256.0		
2.5	70	Retrieval	.028	13	6.9	17,284	14
		Control	1.710	742	1,185.5		
2.5	80	Retrieval	.023	10	0.0	5,067	14
		Control	1.695	705	1,123.8		
2.5	90	Retrieval	.023	10	0.0	4,875	14
		Control	1.685	710	1,089.8		
2.5	100	Retrieval	.023	10	0.0	3,505	14
		Control	1.597	620	869.3		
2.5	200	Retrieval	.023	10	0.0	266	14
		Control	1.398	250	8.0		

or modified page lists. However, as the number of simultaneous VAX users increases, the number of TOTAL DBMS pages resident in the free or modified page lists will decrease, and the number of page faults requiring disk reads will increase - causing response times to increase. This is illustrated in Table VI-2 with memory configurations of .75 megabytes and working set sizes of 80 and 90 pages. The higher number of page fault reads increased the response time of the control commands by approximately .75 seconds even though the actual number of page faults decreased. (Note: All the performance parameter values shown in Table VI-2 were obtained through the instrumentation utility except for the number of page fault reads. The SYS$GETJPI System Services Routine is not currently able to monitor the number of page faults reads; however, this data can be obtained from the summary files produced by the accounting utility (ref. 21:Chapter 21 and Table E-2).

3. As working set and main memory size increased, the number of page faults requiring page reads from disk approached zero and stabilized. This relationship is attributable to the page management characteristics of the VMS operating system. Additional main memory allows the size of the free and modified page lists to increase, thereby, pages not currently contained in the working set remain in main memory - reducing the

number of required page reads when a page fault occurs. Therefore, increasing the size of main memory can alleviate performance problems caused by an excessive page read rate.

4. The main memory configuration of .5 megabytes required a working set size of 50 pages. Attempts to increase the working set caused VMS processes to be swapped out and the TOTAL DBMS would not load.

Test Three. The goal of this test was to determine the overhead effects of the instrumentation utility. The same DBMS applications program and set of DML commands used in Test Two were used for this test. To obtain performance parameter comparisons, the program was executed with the instrumentation utility included in it and without the instrumentation utility included in it. Table VI-3 contains the results of this test, and this data shows the instrumentation utility increased the program's overall execution time by 19.07 seconds (16 percent), increased the CPU time by 9.21 seconds (66 percent), and increased the number of direct I/Os by 25 (100 percent).

Evaluating the overhead, the additional 9.21 seconds of CPU time is primarily caused by the SYS$GETJPI routines and can not be reduced. However, the other 10 seconds of overhead is attributable to the

Table VI-3

Instrumentation Utility Overhead

PERFORMANCE PARAMETER	WITH UTILITY	WITHOUT UTILTY
EXECUTION TIME	2 min, 16.66 sec	1 min, 57.59 sec
CPU TIME	23.20 sec	13.99 sec
DIRECT I/O COUNT	50	25

increase in the number of direct I/Os. This increase is caused by the writing of the data file generated by the instrumentation utility, and it can be reduced by increasing the blocking factor of the file.

When the overhead is factored over the entire set of 1,724 DML commands, the overhead per DML statement is 11 milliseconds (msec) of additional response time. This overhead is constant regardless of the type of DML statement being measured; however, the impact of the overhead is weighted by the type of DML statement where the statements with the fastest response times are affected the most. Using the average values in Table VI-1, an 11 msec increase corresponds to a 50 percent increase in the response time for retrieval commands, a 16 percent increase for storage commands, and a less than one percent increase for control commands. In a program with a closely balanced mix of retrieval and storage commands, the instrumentation utility can be expected to increase average DML response time by approximately 25 to 30 percent; however, this should not affect the "visible" response time of each DML statement. Therefore, overall instrumentation utility overhead is considered minimal as it does not adversely affect DBMS, VAX 11/780, or applications program performance.

Conclusions

On the basis of the work performed during this study, the following conclusions on monitoring DBMS performance are made:

1. Existing performance tools for a general purpose computer system can provide valuable information for evaluating high level DBMS performance. However, detailed performance information at the DML statement level requires a more specialized type of performance tool, such as the instrumentation utility. Inversely, the instrumentation utility is not able to provide a high level view of DBMS performance. It is only suited for detailed types of performance information. Therefore, the use of a combination of different performance tools was a necessity for obtaining all required performance information. A single performance tool would not have satisfied the performance monitoring objectives of all users.

2. The generalized design for a DBMS performance monitor proved to be valid for the VAX 11/780 and TOTAL DBMS. Additionally, it is valid for all other types of DBMSs available for the VAX 11/780 computer and VMS operating system. Also, the instrumentation utility showed it is possible to monitor a DBMS at the DML statement level regardless of the data model it is based on.

3. The top-down design techniques of SADT diagrams, stepwise refinement, and iterative enhancement helped to simplify the task of solving the problem attacked by this study. The results of these techniques simplified the program development stage, and they have provided an exceptional level of documentation. However, the preparation of the documentation is a substantial task, and it is as prone to underestimation of time requirements as is the actual software development!

4. A tool such as the instrumentation utility is relatively straightforward to develop when the operating system provides a set of "hooks" for instrumentation. The System Services Library (ref. 18:124-132) of the VAX/VMS operating system, is an excellent example of a hopefully continuing trend to make computer systems easier to measure.

5. A part of DBMS design requirements should be performance monitoring capabilities. While the

instrumentation utility is a convienient and easy to use tool, it would be better if it were designed as a part of the DBMS. For example, it could be made an extension to the DBMS Log Facility.

6. The DEC extensions to the PASCAL language can be misleading. It is possible to use their PASCAL language to access low-level operating system services via the system services library; however, there is a point at which it becomes cumbersome to do so. Some of the system services library routines require data structures with a precise, low level definition and word/byte boundary constraints. The definition of these structures in a PASCAL program is tedious, prone to error, and confusing (especially to those expected to maintain the programs). Therefore, extreme care must be taken when choosing a programming language for a design requiring the functions provided by the System Services Library. Even though a higher learning curve is normally associated with it, the MACRO-11 Assembly Language may be a better choice than PASCAL.

Recommendations

Based on the results of this study and the observations made during it, the following recommendations for further study and development are made:

1. The remaining pieces of the DBMS performance monitor should be completed. The remaining development work is outlined below by individual program.

User Interface -

a. Allow specific or generalized subsets of performance parameters to be selected and measured.

b. Allow the status of a measurement session to be displayed and modified.

c. Allow the consolidated measurement report to be displayed on a terminal.

Data Analysis Program -

a. Compute statistics on the number of accesses to each data base and the individual files within the data base.

b. Generate information suitable for constructing and evaluating queueing models.

Measurement Report Program - Needs to be developed and tested.

2. The measurement capabilities of the instrumentation utility should be further enhanced. This utility currently monitors a pre-selected, minimum set of performance parameters. Its capabilities can be expanded to include additional performance parameters. This

increases its generality by allowing users to select additional performance parameters to be measured or to default to the minimum subset. This capability will allow users to better design and measure specific performance tests and experiments.

3. A benchmark data base and set of queries should be developed. Using the benchmark to generate example data bases and translating the queries to appropriate DML statements, the performance of different types of DBMSs could be compared using the DBMS performance monitor.

4. A suitable statistical package should be obtained for the VAX computer, such as SPSS-X (ref. 22:83) or an updated version of the Haessle STAT Package (ref. 40:). Since the instrumentation utility records the arrival and completion times of DML statements, arrival and service time distrubutions can be calculated for use in queueing models. This would make it possible to conduct experiments for determining if different types of DBMSs fit different models or if there appears to be a model which can be universally applied to all types of DBMSs. Additionally, the statistical package would be helpful in deriving regression models used for explaining the response time of DML statements.

Bibliograpy

1. Aho, Alfred V., et al. Data Structures and Algorithms. Reading: Addison-Wesley Publishing Company, 1983.

2. Atre, S. DATA BASE: Structured Techniques for Design, Performance, and Management. New York: John Wiley and Sons, Inc., 1980.

3. Badre, Albert N. "Designing the Human - Computer Interface," ACM SIGCSE Bulletin, 14: 41-44 (September 1982).

4. Basili, Victor R. and Turner, Albert J. "Iterative Enhancement: A Practical Technique for Software Development," IEEE Tutorial on Structured Programming, Catalog No. 75CH1049-6: 121-127 (1977).

5. Bell, T. E., et al. Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort. Report R-549-1-PR. Santa Monica, California: Rand Corporation, November 1972.

6. Bray, Olin H. and Freeman, Harvey A. Data Base Computers. Lexington: D.C. Heath and Company, 1979.

7. Brownsmith, Joseph D. A Methodology for the Performance Evaluation of Data Base Systems: An Extension of the IPSS Methodology. PhD Dissertation. Columbus, Ohio: Ohio State University, 1979. (NASA 79N34080).

8. Cincom Systems Inc. Publication Number P10-0002-01. Canada: Cincom Systems Inc., 1979.

9. Cincom Systems Inc. Data Base Administrator's Guide for VAX-11 Systems (Publication Number P10-0001-02). Canada: Cincom Systems Inc., 1981.

10. Datapro Research Corporation. Datapro 70 and Reports on Minicomputers. Delran: Datapro Research Corporation, 1983.

11. Date, C. J. An Introduction to Database Systems (Third Edition). Reading: Addison-Wesley Publishing Company, 1982.

12. Dearnly, P. "Monitoring Database System Performance," Computer Journal, 21: 15-19 (January 1978).

13. DeLutis, Thomas G. A Methodology for the Performance Evaluation of Information Processing Systems. Technical Report OSU-CISRC-TR-77-2. Columbus, Ohio: Ohio State University, March 1977.

14. DeMarco, Tom. Structured Analysis and System Specification. New York: Yourdan Press, 1978.

15. Digital Equipment Corporation. Hardware Handbook. Maynard, Ma.: DEC. 1980.

16. Digital Equipment Corporation. Software Product Description, VAX-11 SPM. Maynard, Ma.: DEC, 1983.

17. Digital Equipment Corporation. VAX/VMS Summary Description and Glossary. Maynard, Ma.: DEC, 1982.

18. Digital Equipment Corporation. VAX/VMS System Services Reference Manual. Maynard, Ma.: DEC, 1982.

19. Digital Equipment Corporation. VAX-11 Pascal User's Guide. Maynard, Ma.: DEC, 1981.

20. Digital Equipment Corporation. VAX-11 Run-Time Library User's Guide. Maynard, Ma.: DEC, 1982.

21. Digital Equipment Corporation. VAX-11 Utilities Reference Manual. Maynard, Ma.: DEC, 1982.

22. Engineering Systems Group. Software Referral Catalog. Marlboro, Ma.: Digital Equipment Corporation, 1983.

23. Ferrari, Domenico and Liu, Mark. "A General-Purpose Software Measurement Tool," Software-Practice and Experience, 5: 181-192 (1975).

24. Ferrari, Domenico. Computer Systems Performance Evaluation. Englewood Cliffs: Prentice-Hall, Inc., 1978.

25. Gilpin, Eugene C. Jr. Development of Computer Performance Evaluation Tools for VAX-11/780 Computers. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1982.

26. Hansen, Per Brinch. Operating System Principles. Englewood Cliffs: Prentice-Hall Inc., 1973.

27. Hartrum, Thomas C. Lecture materials and Computer Performance Evaluation Notes distributed in EE6.52, Computer Performance Measurement and Evaluation. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1983.

28. Hawthorn, Paula B. Evaluation and Enhancement of the Performance of Relational Database Management Systems. PhD Dissertation. Berkley, California: University of California, 1979. (NASA 80N28233).

29. Hawthorn, Paula and Stonebraker, Micheal. "Performance Analysis of a Relational Data Base Management System," Proceedings of ACM-SIGMOD 1979 International Conference on Management of Data, 1-12 (1979).

30. Horowitz, Ellis and Sahni, Sartaj. Fundamentals of Data Structures. Rockville: Computer Science Press, Inc., 1982.

31. Houstis, Catherine E. "Performance Evaluation of a Data Base System," South Eastern Conference on Systems Theory, 251-255 (1980).

32. Jacob, Robert J. K. "Using Formal Specifications in the Design of a Human - Computer Interface," Communications of the ACM, 26: 259-264 (April 1983).

33. Madnick, Stuart E. and Donovan, John J. Operating Systems. New York: McGraw-Hill Book Company, 1974.

34. Maryanski, Fred J. "Backend Database Systems," Computing Surveys, 12: 3-24 (March 1980).

35. Merrill, Hebert W. A Comprehensive Approach to the Performance Measurement and Evaluation of Large-Scale Computer Systems. PhD Dissertation. Champaign, Illinois: University of Illinois at Urbana-Champaign, 1979. (NASA 80N18765).

36. Myers, Glenford J. Advances in Computer Architecture (Second Edition). New York: John Wiley and Sons Inc., 1982.

37. Myers, Glenford J. The Art of Software Testing. New York: John Wiley and Sons Inc., 1979.

38. Norman, Donald A. "Design Rules Based on Analyses of Human Error," Communications of the ACM, 26: 254-258 (April 1983).

39. Oliver, N. N. and Joyce, John D. "Performance Monitor for a Relational Information System," Proceedings of ACM 76 Annual Conference, 329-333 (1976).

40. Palmer, Lars. Haessle STAT Package Users Manual (Version 9A.00). Molndal, Sweden. 1981.

41. Pressman, Roger S. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill Book Company, 1982.

42. Rodriguez-Rosell, Juan and Hildebrand, David. A Framework For Evaluation Of Data Base Systems. IBM Technical Report RJ 1587. San Jose, California: IBM Research Laboratory, May 23, 1975.

43. Shaw, Alan C. The Logical Design of Operating Systems. Englewood Cliffs: Prentice-Hall Inc., 1974.

44. Simpson, Henry. "A Human-Factors Style Guide for Program Design," Byte, 7: 108-132 (April 1982).

45. Smith, Hugh and Green, Thomas. Human Interaction With Computers. New York: Academic Press, 1980.

46. Svobodova, Liba. Computer Performance Measurement and Evaluation Methods: Analysis and Applications. New York: American Elsevier Publishing Company, Inc., 1976.

47. Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs: Prentice-Hall Inc., 1981.

48. Tuel, William G. Jr. and Rodrigues-Rosell, Juan. A Methodology for Evaluation of Data Base Systems. IBM Technical Report RJ 1668. San Jose, California: IBM Research Laboratory, October 15, 1975.

49. Ullman, Jeffrey D. Principles of Database Systems. Potomac: Computer Science Press, Inc., 1980.

50. Weinberg, Victor. Structured Analysis. New York: Yourdan Press, 1980.

51. Wiederhold, Gio. Database Design. New York: McGraw-Hill Book Company, 1977.

52. Wong, Patrick M. K. Performance Evaluation of Data Base Systems. Ann Arbor: UMI Research Press, 1981.

53. Zelkowitz, Marvin V., et al. Principles of Software Engineering and Design. Englewood Cliffs: Prentice-Hall Inc., 1979.

Vita

Paul Dennis Bailor was born in Lebanon, Pennsylvania on July 9, 1953. He graduated from high school in 1971 and enlisted in the United States Air Force in May, 1972. He received an Air Force Reserve Officers Training Corps scholarship in July 1975, and he entered the University of Maryland, College Park, Maryland in January 1976. He graduated Magna Cum Laude with the degree of Bachelor of Science in Computer Science in December 1978. After graduation, he served as a Department of Defense Computer Programmer at Headquarters Military Enlistment Processing Command, Fort Sheridan, Illinois. During this assignment, he was in charge of developing and implementing a network of mini-computers for the 68 Military Enlistment Processing Stations located throughout the United States and Puerto Rico. He entered the Air Force Institute of Technology in June 1982.

Permanent Address: 1537 Sand Hill Road
Lebanon, Pennsylvania

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/EE/83D-2

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*
AFIT/ENG

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE *(Include Security Classification)*
See Box 19

12. PERSONAL AUTHOR(S)
Bailor, Paul D., B.S., Capt, USAF

13a. TYPE OF REPORT
MS Thesis

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Yr., Mo., Day)*
1983 December

15. PAGE COUNT
506

16. SUPPLEMENTARY NOTATION

Approved for public release: IAW AFR 190-17.
[illegible]
LYNN E. WOLAVER 7 Feb 84
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

17. COSATI CODES

FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Data Bases, Data Base Management System, Data Management, DBMS, Computer Performance Evaluation Performance(Engineering), Monitors

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: DEVELOPMENT OF A DATA BASE MANAGEMENT SYSTEM
PERFORMANCE MONITOR

Thesis Advisor: Dr. Gary B. Lamont, Professor, EE Department

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED [✓] SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Dr. Gary B. Lamont, EE Department

22b. TELEPHONE NUMBER *(Include Area Code)*
513-255-3450

22c. OFFICE SYMBOL
AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

This study focuses on the problem of evaluating the performance of a Data Base Management System (DBMS). In this study, DBMS performance evaluation is treated as a subset of computer performance evaluation, and in doing this, the performance parameters unique to a DBMS were developed and merged with the performance parameters associated with a general purpose computer system.

Based on this approach, a generalized design for a DBMS performance monitor was developed. This design emphasizes the use of existing performance tools such as software monitors and accounting packages, and it takes the performance monitoring requirements of different types of DBMS users into consideration. Additionally, the design is applicable to any type of DBMS regardless of the underlying data model.

The generalized design was implemented on a VAX 11/780 computer for the TOTAL DBMS. The results of the implementation showed the generalized design was viable and capable of measuring many different types of DBMSs. However, existing performance tools were only capable of providing a high level picture of DBMS performance. A specialized tool called an instrumentation utility had to be developed to gather detailed performance information.